ANALYSIS OF TOPOLOGICAL CHAOS IN GHOST ROD MIXING AT FINITE REYNOLDS NUMBERS USING SPECTRAL METHODS

A Thesis

by

PRADEEP CHANDRAKANT RAO

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2009

Major Subject: Mechanical Engineering

ANALYSIS OF TOPOLOGICAL CHAOS IN GHOST ROD MIXING AT FINITE REYNOLDS NUMBERS USING SPECTRAL METHODS

A Thesis

by

PRADEEP CHANDRAKANT RAO

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Andrew T. Duggleby
Committee Members,	N. K. Anand
	Hamn-Ching Chen
Head of Department,	Dennis O'Neil

December 2009

Major Subject: Mechanical Engineering

ABSTRACT

Analysis of Topological Chaos in Ghost Rod Mixing
at Finite Reynolds Numbers
Using Spectral Methods. (December 2009)
Pradeep Chandrakant Rao, B.E., University of Mumbai
Chair of Advisory Committee: Dr. Andrew Duggleby

The effect of finite Reynolds numbers on chaotic advection is investigated for two dimensional lid-driven cavity flows that exhibit topological chaos in the creeping flow regime. The emphasis in this endeavor is to study how the inertial effects present due to small, but non-zero, Reynolds number influence the efficacy of mixing. A spectral method code based on the Fourier-Chebyshev method for two-dimensional flows is developed to solve the Navier-Stokes and species transport equations. The high sensitivity to initial conditions and the exponentional growth of errors in chaotic flows necessitate an accurate solution of the flow variables, which is provided by the exponentially convergent spectral methods. Using the spectral coefficients of the basis functions as solved through the conservation equations, exponentially accurate values of velocity everywhere in the flow domain are obtained as required for the Lagrangian particle tracking. Techniques such as Poincaré maps, the stirring index based on the box counting method, and the tracking of passive scalars in the flow are used to analyse the topological chaos and quantify the mixing efficiency.

Dedicated to the people who matter to me the most: my parents and my fiancée

ACKNOWLEDGMENTS

I should like to thank my advisor, Dr. Andrew Duggleby, for taking me under his tutelage and for always believing in my abilities. I would also like to thank Dr. Mark Stremler of ViginiaTech, for his input on lid-driven cavity flow.

I shall forever cherish the friendship, support and guidance of Yuval Doron, Markus Schwænen, Shriram Jagannathan, Dallas Potz-Nielson, Marshall Whitney and Jared Broz, all student members of FT2L.

NOMENCLATURE

D	mass diffusivity
M	mixing index
P	dimensionless pressure
Pe	Peclet number, $= U_{max}h/D$
Re	Reynolds number, $= U_{max}h/\nu$
Sc	Schmidt number, $= \nu/D$
$T_k(y)$	k^{th} Chebyshev polynomial
U_{max}	maximum value of u at the boundary
$\underline{k}$	unit vector in z direction
$\underline{u}$	dimensionless velocity vector
h	half height of flow domain
u	x component of dimensionless velocity
v	y component of dimensionless velocity
x	dimensionless horizontal coordinate
y	dimensionless vertical coordinate
δ_{kl}	Kronecker delta
ϵ	stirring index

ν	kinematic viscosity
ω	z component of dimensionless vorticity
ψ	stream function
θ	dimensionless concentration
$\underline{\omega}$	dimensionless vorticity vector

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION AND LITERATURE REVIEW

A. Chaotic Advection

Turbulence plays a major role in the mixing of fluids, due to the presence of a large range of scales of fluid motion that enhance species transport and diffusion. In the absence of turbulence, for flows at extremely small Reynolds numbers, the only mechanism driving mixing is diffusion, which is a slow process. A way to speed up this process is to "stir" the fluid in a manner to increase the interface accross which diffusion occurs, thereby increasing the mixing rate. By ensuring that the trajectories of particles advected with the flow be chaotic, an exponential stretching rate of the interface can be achieved, leading to quick and efficient mixing. There exist many real world flow regimes or applications where turbulence is either not possible to achieve, or is undesirable. In micro flows, viscous forces dominate, and mixing only through diffusion can take long times in spite of the small length scales, due to poor diffusivities. In typical biofluidic appplications, turbulence can cause undesirable large strains on embedded macromolecules or biological species, which makes mixing in the laminar regime an attractive alternative. Chaotic mixing also finds application in materials processing, for example in the production of multi-layerd polymer films [1].

The term chaotic advection was introduced by Aref [2], who showed that using simple, laminar, time-dependant flow patterns, one can achieve chaotic particle trajectories. This leads to enhanced mixing through exponential stretching and folding of the fluid. In [3], Jones et al. showed chaotic advection developed by laminar flow

---

The journal model is *IEEE Transactions on Automatic Control.*

in a twisted pipe. Aref et al. highlighted the role of vorticity and vortices in chaos in [4]. It was also shown in [5] that the reversibility of advection and irreversibility of diffusion in Stokes flows can be combined to effectively separate substances with close diffusivities. Boyland et al. [6] showed how topological chaos can be achieved by the motion of 3 or more stirres in 2 dimensional flows. In [7] it was shown how vortices can act as stirrers. For a detailed description of the nature and origins of chaotic advection, one can refer Aref's paper based on his 2000 Otto Laporte Memorial Lecture [8] or the review paper on the foundations of chaotic mixing by Wiggins and Ottino [9].

Most of the early focus in the study of chaotic advection has been on Stokes flows. Such flows are determined completely by the boundary conditions and therefore exact analytical solutions can be found for them. The exact solutions enable the accurate tracking of particle trajectories in the flow. The Stokes flow assumption however does not take into account the inertial effects in fluid motion that are present in all real flows. Therefore, the predictions of Stokes flow solutions are valid only for small, initial times, since the chaotic motion magnifies errors at an exponential rate. It is therefore of value to study how inertial effects influence mixing. Dutta and Chevray observed that inertial effects significantly enhanced mixing in the annular flow between two eccentric cylinders [10]. Hobbs and Muzzio found that for a Kenics static mixer, the formation of non-chaotic islands in the flow for Re>10 lead to a loss in mixing efficiency [11]. Clifford et al. studied inertial effects in a simple planetary mixer and observed shrinking of non-chaotic islands with increasing Reynolds number [12]. They concluded that the best mixing protocol depends strongly on the Reynolds number of operation of the mixing device. Wang et al. found good agreement with Stokes flow predictions for 2D cavity flows for Re$\leq$10 [13]. From these studies one can conclude that, though Stokes flow solutions are good at predicting which

flow configurations can achieve chaotic advection, and thereby efficient mixing, it is necessary to study these flows at finite Reynolds number for the simulation and optimisation of real world mixers. It is interesting to note, that most numerical studies of chaotic advection at finite Reynolds number have been done using traditional finite volume methods that achieve only 2nd order accuracy.

In the current numerical study, the effect of inertia on chaotic advection is investigated for two dimensional lid-driven cavity flows similar to those studied in [14], that exhibit topological chaos in the creeping flow regime. The governing equations for the flow domain are solved by using a spectral method algorithm.

B. Spectral Methods

What distinguishes spectral methods from finite element or finite difference methods, is the choice of trial or basis functions [15]. The trial basis functions for classical spectral methods, like the one used in this study, on a single tensor-product domain are global, infinitely differentiable and nearly orthogonal [15]. Due to this, the global error decreases exponentially with the number of degrees of freedom, as evidenced in Figure 1. Lagrangian particle tracking is an important tool required in the analysis of mixing and topological chaos. Chaotic flows, as already mentioned, are characterised by high sensitivity to initial conditions, as a result of which, is errors grow at an exponential rate. The main motivation behind writing own code, lies in the ability to use the spectral coefficients of velocity obtained from solving the conservation equations, to get spectrally accurate values of velocity at any given point in the flow field, as required for particle tracking; without having to resort to interpolation between grid points, which leads to a general loss of accuracy. Thus, incorporation of the algorithms to evaluate mixing efficiency and topological chaos in the spectral

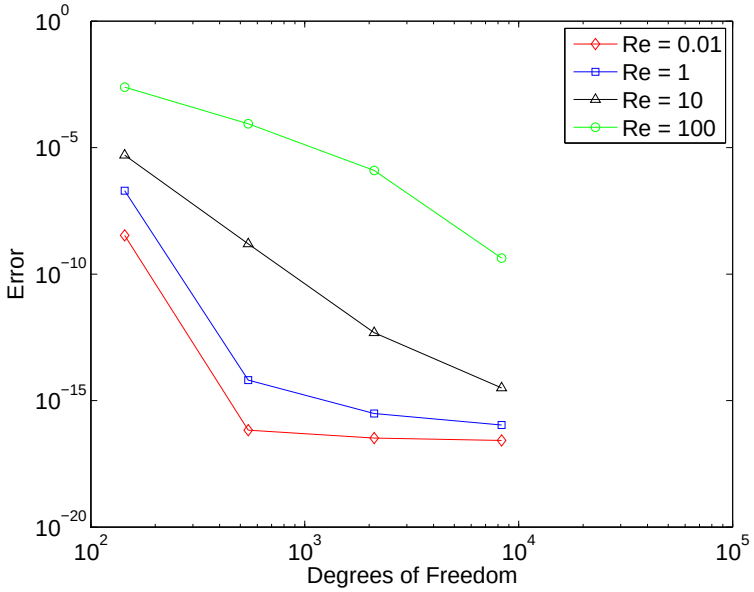


Fig. 1. Spectral convergence of stream function, for $Re = 0.01$, $Re = 1$, $Re = 10$ and $Re = 100$.

method code, makes it possible for these tools to take advantage of the exponential convergence afforded by the use of spectral methods.

CHAPTER II

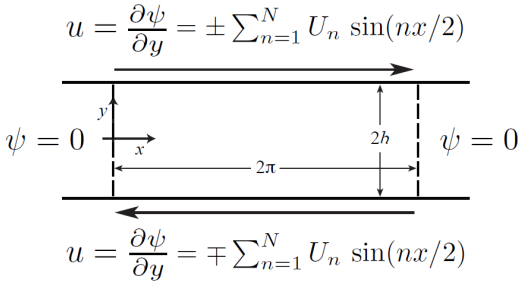
PROBLEM FORMULATION AND MATHEMATICAL PRELIMINARIES

A. Description of the Flow Domain

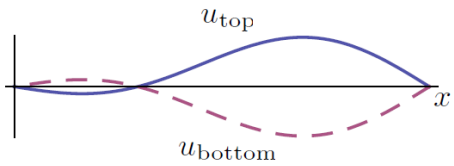
Chaotic advection has been achieved in a variety of flow configurations, using patterned walls, electro-osmotic effects etc. Boyland et al. [6] showed how topological chaos can be achieved by the motion of 3 or more stirres in 2 dimensional flows. It was demonstrated, how, for 3 stirres, there are two intrinsically different ways in which they can be moved about. One of them is topologically trivial and the other which is akin to the manner of braiding hair into plats, is topologically profound. In [7] it was shown by the same authors, how the braiding motion of 3 stirres can be achieved by vortices in place of the stirrers.

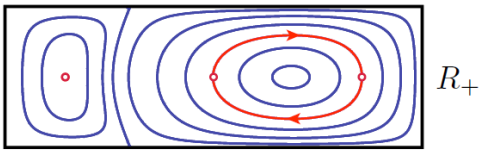
It is this same principle that is used to generate topological chaos in the current study. The flow configuration used alongwith it's analytical solution, was developed and provided for this study by Mark A. Stremler [16], who is a faculty member in the Engineering Science & Mechanics Department at the Virginia Polytechnic Institute & State University.

Consider a 2 dimensional flow domain, $-\frac{\pi}{3} \leq y \leq \frac{\pi}{3}$ which is periodic in the x-direction with a period of 4π. Each period can be divided into two halves of length 2π each. Each half contains two counter-rotating cells, that are generated by the boundary conditions for u at $y = \pm\frac{\pi}{3}$. The purpose generating the counter-rotating cells is to interchange the position of three periodic points in the flows in every half cycle, namely R+ and L- as depicted in Figure 2. These elleptic points behave like "ghost-rods" [7] and generate topological chaos by stirring the flow. As seen in Figure 3 when one half of the domain undergoes the R+ cycle, the other half undergoes the

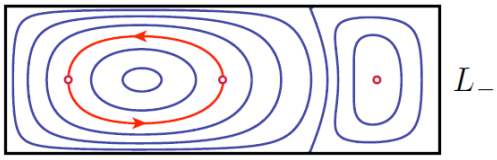


(a)



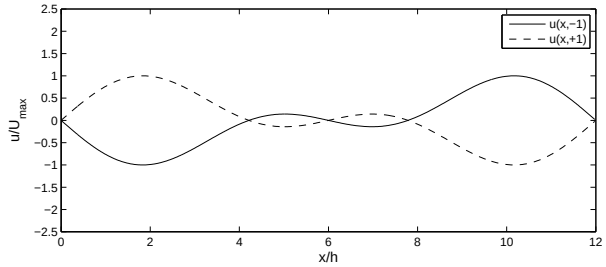


(b)

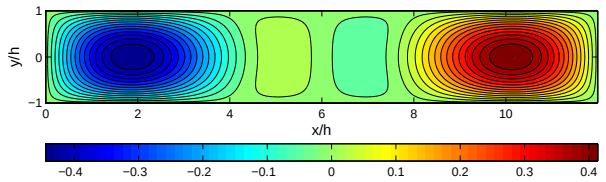


(c)

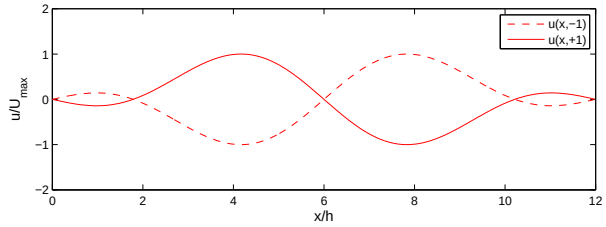
Fig. 2. Flow domain and boundary conditions. Above figures used from the presentation titled 'Almost invariant sets as "ghost rods" for fluid stirring' by, Mark A. Stremler, Shane D. Ross, Piyush Grover and Pankaj Kumar. Figures (a), (b) and (c) show half the flow domain. Figure (a) shows the boundary conditions. Figures (b) and (c) show the R+ and the L- braiding cycles using ghost rods, respectively



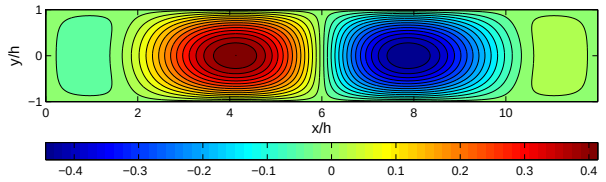
(a)



(b)



(c)



(d)

Fig. 3. Figures (a) and (b) show boundary conditions for u velocity and contours of stream function at the end of the R+ half cycle, and (c) and (d) show boundary conditions for u velocity and contours of stream function at the end of the L-half cycle respectively for Re = 0.01.

L- cycle. It is therefore, sufficient to study the domain for $0 \leq x \leq 2\pi$ to determine the characteristics of the flow.

The Stokes flow, analytical solution for this configuration is given as follows: Assume a stream function of the form

$$\begin{aligned} \psi(x,y) &= \sum_{n=1}^{N} U_n \psi_n(x,y) \\ &= \sum_{n=1}^{N} U_n C_n f_n(y) \sin(\frac{n\pi x}{2a}) \end{aligned} \tag{2.1}$$

The model equation for Stokes flow is given by

$$\nabla^4 \psi_n = 0, \tag{2.2}$$

which implies

$$f_n''''(y) - 2(\frac{n\pi}{2a})^2 f''(y) + (\frac{n\pi}{2a})^4 f_n(y) = 0 \tag{2.3}$$

$$f_n(\pm h) = 0, \quad C_n f_n'(\pm h) = \pm 1, \tag{2.4}$$

The solution to (2.3) is given by

$$f_n(y) = y\cosh(\frac{n\pi h}{2a})\sinh(\frac{n\pi y}{2a}) - h\sinh(\frac{n\pi h}{2a})\cosh(\frac{n\pi y}{2a}) \tag{2.5}$$

$$C_n = \frac{2}{\sinh(\frac{n\pi h}{a}) + \frac{n\pi b}{a}} \tag{2.6}$$

For the case of interest, we take $N = 2$, $a = \pi$, and $h = \frac{\pi}{3}$.

$$\begin{aligned} \psi(x,y) &= U_1 C_1 f_1(y) sin(\frac{\pi x}{2a}) + U_2 C_2 f_2(y) sin(\frac{2\pi x}{2a}) \\ &= U[\sqrt{1-\beta}\psi_1 + \sqrt{\beta}\psi_2] \end{aligned} \tag{2.7}$$

To get chaotic mixing, we need three periodic points along $y = 0$ that will generate the desired braiding motion. Mark Stremler [16] found through analyti-

cal solutions, that this can be achieved by setting $U = 12.973936975838439$, $\beta = 0.414445388932874$. The pulse time, or switching time for every half-cycle for this configuration, $\tau = 0.5$. For the numerical simulations, the velocity is dimensionless by the maximum velocity at the $y = \pm 1$, $U_{max} = 15.981011406369920$, and length is dimensionless by the half height of the domain, $h = \frac{\pi}{3}$. From this point on in this text, x, y, u, v shall signify dimensionless variables.

B. Fourier and Chebyshev Basis Functions

The flow domain under consideration is periodic in the x direction, which enables the use of Fourier approximation in that direction. In the y direction the Chebyshev approximation is applied. The solution sought for any field variable in a domain which is periodic with a period of 2π, is in the form:

$$\phi_N(x, y) = \sum_{l=0}^{\infty} \sum_{k=-\infty}^{\infty} \hat{\phi}_{lk} T_l(y) e^{ikx} \tag{2.8}$$

where $\phi(x, y)$ is any scalar field in the domain and $\phi_N(x, y)$ is its spectral approximation.

Spectral methods make use of orthogonal basis functions or approximation or interpolation functions, as per the inner product defined by the particular basis function. The basic relations for Fourier and Chebyshev basis functions presented in this section are referred from [15] and [17].

1. Fourier Basis Functions

a. Continuous Fourier Expansion

Any function that is periodic in nature can be described by using a Fourier series. Consider the set of functions defined by:

$$\phi_k(x) = e^{ikx} \tag{2.9}$$

These functions are orthogonal as defined by the inner product:

$$\int_0^{2\pi} \phi_k(x)\overline{\phi_l(x)}\,dx = \int_0^{2\pi} e^{ikx}e^{-ilx}\,dx = 2\pi\delta_{kl} = \begin{cases} 0 & \text{if } k \neq l, \\ 2\pi & \text{if } k = l. \end{cases} \tag{2.10}$$

For a complex valued function u defined on the interval $(0,\, 2\pi)$, the Fourier coefficients of u are given by:

$$\hat{u}_k = \frac{1}{2\pi}\int_0^{2\pi} u(x)e^{ikx}\,dx, \qquad k = 0,\, \pm 1,\, \pm 2,\, ... \tag{2.11}$$

Note that, for a real valued function u, $\hat{u}_k = \overline{\hat{u}_k}$. The Fourier series of a function is defined as

$$Su = \sum_{k=-\infty}^{\infty} \hat{u}_k\phi_k \tag{2.12}$$

b. Discrete Fourier Expansion

The discrete Fourier expansion is an interpolation function defined for a set of N points, (where N is any positive integer) given by

$$x_j = \frac{2\pi j}{N}, \qquad j = 0,\, 1,...,\, N-1 \tag{2.13}$$

The discrete Fourier coefficients for a complex valued function u are then given by

$$\hat{u}_k = \frac{1}{N}\sum_{j=0}^{N-1} u(x_j)e^{-ikx}, \qquad k = -N/2,..., N/2-1. \tag{2.14}$$

From the orthogonality relation:

$$\frac{1}{N}\sum_{j=0} N-1e^{-ipx_j} = \begin{cases} 1 & \text{if } p = Nm,\ m = 0, \pm1, \pm2,..., \\ 0 & \text{otherwise,} \end{cases} \tag{2.15}$$

we have the inversion formula

$$u(x_j) = \sum_{k=-N/2}^{N/2-1} \hat{u}_k e^{ikx_j}, \qquad j = 0,\ 1,...,\ N-1 \tag{2.16}$$

Thereby, the discrete Fourier interpolant function $u_K(x)$ of $u(x)$ defined over the domain $[0, 2\pi]$ is given by the relation

$$u_K(x) = \sum_{k=-N/2}^{N/2-1} \hat{u}_k e^{ikx} \tag{2.17}$$

The function $u_K(x)$ is defined by its values at the N interpolation points, namely $x_j = \frac{2\pi j}{N},\ j = 0,\ 1,...,\ N-1$ in the physical space, and by the values of the N coefficients of its Fourier interpolation, namely $\hat{u}_k,\ k = -N/2,...,\ N/2-1$, in the wave space, where k is the wave number.

The first derivative of the Fourier interpolant function u_K is given by

$$\frac{du_K}{dx} = \sum_{k=-N/2}^{N/2-1} (ik)\hat{u}_k e^{ikx} \tag{2.18}$$

Similarly, the second derivative of the Fourier interpolant function u_K is given by

$$\frac{d^2u_K}{dx^2} = \sum_{k=-N/2}^{N/2-1} (-k^2)\hat{u}_k e^{ikx} \tag{2.19}$$

2. Chebyshev Basis Functions

The Chebyshev polynomials of the first kind are given by the relation,

$$T_k(x) = cos(k \cos^{-1} x), \quad -1 \leq x \leq 1, \tag{2.20}$$

where $k = 0, 1, ...,$ and therefore, $-1 \leq T_k(x) \leq 1$.

The Chebyshev polynomials of the first kind are orthogonal over the inner product defined by

$$\int_{-1}^{1} \frac{T_k(x)T_l(x)}{\sqrt{1-x^2}}\, dx = \frac{\pi}{2} c_k \delta_{kl} \quad \text{, where} \qquad c_k = \begin{cases} 2 & \text{if } k = 0, \\ 1 & \text{if } k \geq 1. \end{cases} \tag{2.21}$$

The Chebyshev function is defined as an interpolation over $N + 1$ points given by

$$x_j = -\cos(\frac{\pi j}{N}), \qquad j = 0, 1,..., N \tag{2.22}$$

The interpolant function $u_N(x)$ for a real valued function $u(x)$ defined over the domain $[-1, 1]$ is given by

$$u_N(x) = \sum_{l=0}^{N} \hat{u}_l(x) T_l(x) \tag{2.23}$$

where the coefficients of the Chebyshev ploynomials $\hat{u}_l$ are given by the relation

$$\hat{u}_l = \frac{2}{\pi c_l} \int_{-1}^{1} \frac{u(x)T_l(x)}{\sqrt{1-x^2}}\, dx \tag{2.24}$$

The Chebyshev polynomials can be calculated from the power series given by

$$T_l(x) = \frac{l}{2} \sum_{p=0}^{[l/2]} (-1)^l \frac{(l-p-1)!}{p!(l-2p)!} (2x)^{l-2p} \tag{2.25}$$

where $[l/2]$ denotes the the integral part of $l/2$.

Further, the following recursion formula can also be used

$$T_{l+1}(x) = 2xT_l(x) - T_{l-1}(x), \tag{2.26}$$

with $T_0(x) \equiv 1$ and $T_1(x) \equiv x$.

The derivatives of Chebychev interpolant functions can always be expressed as a linear combination of the Chebyshev polynomials themselves

$$u_N^{(p)}(x) = \sum_{j=0}^{N} \hat{u}_l^{(p)} T_l(x) \tag{2.27}$$

where $\hat{u}_l^{(p)}$ are the chebyshev coefficients of the interpolant function of $u_N^{(p)}(x)$. The first-order derivative is given by

$$u'(x) = \sum_{l=0}^{N} \hat{u}_l^{(1)} T_l(x) \tag{2.28}$$

with

$$\hat{u}_l^{(1)} = \frac{2}{c_l} \sum_{\substack{p=l+1 \\ (p+l)\text{odd}}}^{N} p\hat{u}_p, \quad l = 0,..., N-1 \tag{2.29}$$

and $\hat{u}_N = 0$. This can be written in matrix form as

$$\hat{U}^{(1)} = \hat{D}\,\hat{U}, \tag{2.30}$$

where $\hat{U} = (\hat{u}_0,..., \hat{u}_N)^T$, $\hat{U}^{(1)} = (\hat{u}_0^{(1)},..., \hat{u}_N^{(1)})^T$ and $\hat{D}$ is a strictly upper triangular matrix derived from (2.29). The second-order derivative is given by the relation

$$u''(x) = \sum_{l=0}^{N} \hat{u}_l^{(2)} T_l(x) \tag{2.31}$$

with

$$\hat{u}_l^{(2)} = \frac{1}{c_l} \sum_{\substack{p=l+2 \\ (p+l)\text{even}}}^{N} p(p^2 - l^2)\hat{u}_p, \quad l = 0,..., N-2 \tag{2.32}$$

and $\hat{u}_N = \hat{u}_{N-1} = 0$. This can be written in matrix form as

$$\hat{U}^{(2)} = \hat{D}^2 \hat{U}, \tag{2.33}$$

where $\hat{U}^{(2)} = (\hat{u}_0^{(2)}, ..., \hat{u}_N^{(2)})^T$.

C. Conservation Equations

The dimensionless form of the Navier Stokes equation for incompressible fluids is given by

$$\frac{\partial \underline{u}}{\partial t} + \underline{u} \cdot \underline{\nabla}\, \underline{u} = -\underline{\nabla} P + \frac{1}{Re} \nabla^2 \underline{u} \tag{2.34}$$

where, Re is the Reynolds number. The Reynolds number for the current study is defined as $Re = U_{max} h/\nu$, where $Umax$ is the maximum velocity at the boundary, h is the half height of the cavity and ν is the kinematic viscosity of the fluid. Also, the passive scalar transport equation is given by

$$\frac{\partial \theta_n}{\partial t} + \underline{u} \cdot \underline{\nabla}\, \theta_n = \frac{1}{ReSc_n} \nabla^2 \theta_n, \tag{2.35}$$

where θ_n is the nth passive scalar. By taking the curl of (2.34), we can eliminate the pressure term and obtain the vorticity equation, given by:

$$\frac{\partial \underline{\omega}}{\partial t} + \underline{u} \cdot \underline{\nabla}\, \underline{\omega} = \underline{\omega} \cdot \underline{\nabla}\, \underline{u} + \frac{1}{Re} \nabla^2 \underline{\omega} \tag{2.36}$$

Now consider two dimentional flow in Cartesian coordinates, where the x and y coordinates lie in the plane and the z coordinate is perpendicular to it. For such flows, u and v are the non-zero components of $\underline{u}$, and the only non-zero component of $\underline{\omega}$, namely ω, is in the z direction. The vorticity equation (2.36) thus becomes:

$$\frac{\partial \omega}{\partial t} + u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y} = \frac{1}{Re}\left[\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right] \tag{2.37}$$

The two dimensional flow field can be described by a stream function ψ, which enforces a divergence free velocity field by setting,

$$\underline{u} = \underline{\nabla} \times \psi \underline{k} \tag{2.38}$$

whereby we get,

$$u = \frac{\partial \psi}{\partial y}, \qquad v = -\frac{\partial \psi}{\partial x} \tag{2.39}$$

For the flow field under consideration, the vorticity component in the z direction is given by,

$$\omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{2.40}$$

Therefore from (2.39) we have,

$$\omega = -\left(\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2}\right) \tag{2.41}$$

(2.37) and (2.41) are the equations used to model the flow field.

D. Descretisation of Conservation Equations

From (2.8), we have

$$\begin{aligned} \omega_N(x,y) &= \hat{\omega}_{lk} T_l(y) e^{ikAx}, \qquad \psi_N(x,y) = \hat{\psi}_{lk} T_l(y) e^{ikAx} \\ u_N(x,y) &= \hat{u}lk T_l(y) e^{ikAx}, \qquad v_N(x,y) = \hat{v}_{lk} T_l(y) e^{ikAx} \end{aligned} \tag{2.42}$$

where x and y are dimensionless with the half height of the domain giving $-1 \leq y \leq 1$ as required by the chebyshev basis functions, and $A = 2\pi/L$, L being the period of the domain in the x direction. Note that there is an implied double sum over the indices l and k in (2.42)

Substituting field variables by their spectral approximations in (2.37) and using

the relations for derivatives given in b and 2 we get

$$\frac{\partial\hat{\omega}_{lk}}{\partial t}T_l(y)e^{ikAx} + \hat{u}_{lk}T_l(y)e^{ikAx}(ikA)\hat{\omega}_{lk}T_l(y)e^{ikAx} + \hat{v}_{lk}T_l(y)e^{ikAx}\hat{\omega}^{(1)}_{lk}T_l(y)e^{ikAx} = \frac{1}{Re}[\hat{\omega}^{(2)}_{lk}T_l(y)e^{ikAx} - (Ak)^2\hat{\omega}_{lk}T_l(y)e^{ikAx}] \quad (2.43)$$

By replacing the nonlinear terms in (2.43) with $\hat{Q}_{lk}T_l(y)e^{ikAx}$, we get

$$\frac{\partial\hat{\omega}_{lk}}{\partial t}T_l(y)e^{ikAx} + \hat{Q}_{lk}T_l(y)e^{ikAx} = \frac{1}{Re}[\hat{\omega}^{(2)}_{lk}T_l(y)e^{ikAx} - (Ak)^2\hat{\omega}_{lk}T_l(y)e^{ikAx}] \quad (2.44)$$

By taking an innerproduct as defined by (2.10) on (2.44), we get the set of equations:

$$\frac{\partial\hat{\omega}_{lk}}{\partial t}T_l(y) + \hat{Q}_{lk}T_l(y) = \frac{1}{Re}[\hat{\omega}^{(2)}_{lk}T_l(y) - (Ak)^2\hat{\omega}_{lk}T_l(y)] \quad (2.45)$$

where k takes the integer values from $-N/2$ to $N/2-1$. Thus, a partial differential equation in 2 dimensinos is transformed to a series of discretised ordinary differential equations, one for every Fourier wave number k.

The time stepping is achieved using the semi implicit Adams-Bashforth/Backward-Differentiation Scheme (AB/BDI2). In this method, the linear (diffusive) terms are solved implicitly using a 2nd order Backward-Differentiation scheme, and the nonlinear (convective) terms are treated using an explicit Adams-Bashforth formulation.

For a differential equation:

$$\frac{\partial\underline{u}}{\partial t} = \underline{g}(\underline{u},t) + \underline{f}(\underline{u},t), \quad (2.46)$$

the 2nd order backwards-difference scheme (BDF2) is given by:

$$\underline{u}^{n+1} = \frac{1}{3}[4\underline{u}^n - \underline{u}^{n-1}] + \frac{2}{3}\Delta t[\underline{g}^{n+1} + \underline{f}^{n+1}], \quad (2.47)$$

where $\underline{g}$ represents the linear terms and $\underline{f}$ represents the nonlinear terms in the differential equation. Extrapolating the nonlinear terms to the (n+1)th time iterate

using the Adams-Bashforth scheme, we get:

$$\underline{u}^{n+1} = \frac{1}{3}[4\underline{u}^n - \underline{u}^{n-1}] + \frac{2}{3}\Delta t[\underline{g}^{n+1} + 2\underline{f}^n - \underline{f}^{n-1}] \tag{2.48}$$

Applying the AB/BDI2 scheme to (2.45), we get:

$$\hat{\omega}_{lk}^{n+1} T_l(y) = \frac{1}{3}[4\hat{\omega}_{lk}^n - \hat{\omega}_{lk}^{n-1}]T_l(y) + \frac{2}{3}\delta t[\hat{\omega}_{lk}^{(2)^{n+1}} + 2\hat{Q}_{lk}^n - \hat{Q}_{lk}^{n-1}]T_l(y) \tag{2.49}$$

or,

$$\hat{\omega}_{lk}^{(2)} T_l(y) - \sigma\hat{\omega}_{lk} T_l(y) = \hat{f}_{lk} T_l(y) \tag{2.50}$$

where $\sigma = \frac{3Re}{2\Delta t} - (kA)^2$ and $\hat{f}_{lk} = 2\hat{Q}_{lk}^n - \hat{Q}_{lk}^{n-1}$. (2.50) is solved using the Chebyshev Tau method.

1. Fourier Chebyshev Method: Influence Matrix

The pseudo spectral method used in the current study is the influence matrix method outlined in [17]. Consider a channel that extends infintely in the x direction, and is periodic in this direction with a period of 2π, and $-1 \leq y \leq 1$. We assume that the initial velocity field also periodic. For this flow, we have Direchlet boundary conditions for velocity prescribed at $y = \pm 1$. Since there is no pressure gradient driving the flow, there is no net flow in the channel. Therefore, we have,

$$\psi(x, -1, t) = 0, \qquad \psi(x, 1, t) = 0 \tag{2.51}$$

$$\frac{\partial\psi}{\partial y}(x, -1, t) = u_{\text{-}}(x, t), \qquad \frac{\partial\psi}{\partial y}(x, 1, t) = u_{+}(x, t) \tag{2.52}$$

The truncated Fourier expansions for vorticity and stream function are given by:

$$\omega_K(x,y,t) = \sum_{k=-N/2}^{N/2-1} \hat{\omega}_k(y,t)e^{ikx}$$
$$\psi_K(x,y,t) = \sum_{k=-N/2}^{N/2-1} \hat{\psi}_k(y,t)e^{ikx}$$

Fourier Galerkin equations satisfied by $\hat{\omega}_k$, $\hat{\psi}_k$, $k = -N/2$,..., $N/2-1$ are

$$\frac{\partial \hat{\omega}_k}{\partial t} - \frac{1}{Re}(\frac{\partial^2 \hat{\omega}_k}{\partial y^2} - k^2\hat{\omega}_k) = \hat{F}_k \ \text{ in } \ -1 \le y \le 1 \tag{2.53}$$

$$\frac{\partial^2 \hat{\psi}_k}{\partial y^2} - k^2\hat{\psi}_k + \hat{\omega}_k = 0 \ \text{ in } \ -1 \le y \le 1 \tag{2.54}$$

$$\hat{\psi}_k = \hat{g}_{-,k}, \quad \frac{\partial \hat{\psi}_k}{\partial y} = \hat{h}_{-,k} \ \text{ at } y = -1 \tag{2.55}$$

$$\hat{\psi}_k = \hat{g}_{+,k}, \quad \frac{\partial \hat{\psi}_k}{\partial y} = \hat{h}_{+,k} \ \text{ at } y = 1 \tag{2.56}$$

The N one dimentional problems in the y direction are solved with the Chebyshev Tau method, where $\hat{\omega}_{kN}$ and $\hat{\psi}_{kN}$ are the Chebyshev approximations of $\hat{\omega}_k$ and $\hat{\psi}_k$.

$$\check{\omega}'' - \sigma\check{\omega} = f, \qquad -1 < y < 1 \tag{2.57}$$

$$\check{\psi}'' - k^2\check{\psi} + \check{\omega} = 0, \qquad -1 < y < 1 \tag{2.58}$$

$$\check{\psi}(-1) = g_-, \quad \check{\psi}(1) = g_+, \tag{2.59}$$

$$\check{\psi}'(-1) = h_-, \quad \check{\psi}'(+1) = h_+, \tag{2.60}$$

where $\check{\omega} = \hat{\omega}_k^{n+1}$etc. The solution to $\check{\omega}$ and $\check{\psi}$ is sought through the decomposotion:

$$\check{\omega} = \tilde{\omega} + \bar{\omega}, \qquad \check{\psi} = \tilde{\psi} + \bar{\psi} \tag{2.61}$$

$$\bar{\omega} = \xi_1\bar{\omega}_1 + \xi_2\bar{\omega}_2, \quad \bar{\psi} = \xi_1\bar{\psi}_1 + \xi_2\bar{\psi}_2 \tag{2.62}$$

where $\tilde{\omega}$ and $\tilde{\psi}$ satisfy the problem defined by

$$\tilde{\omega}'' - \sigma\tilde{\omega} = f, \quad -1 < y < 1 \tag{2.63}$$

$$\tilde{\omega}(-1) = 0, \quad \tilde{\omega}(1) = 0, \tag{2.64}$$

$$\tilde{\psi}'' - k^2\tilde{\psi} = -\tilde{\omega}, \quad -1 < y < 1 \tag{2.65}$$

$$\tilde{\psi}(-1) = g_-, \quad \tilde{\psi}(+1) = g_+, \tag{2.66}$$

$$\tilde{\psi}'(-1) = h_-, \quad \tilde{\psi}'(+1) = h_+, \tag{2.67}$$

where the elementary solutions $(\bar{\omega}_l, \bar{\psi}_l)$, $l = 1, 2$, satisfy the following equations:

$$\bar{\omega}_1'' - \sigma\bar{\omega}_1 = f, \quad -1 < y < 1 \tag{2.68}$$

$$\bar{\omega}_1(-1) = 1, \quad \bar{\psi}_1(1) = 0, \tag{2.69}$$

$$\bar{\psi}_1'' - k^2\bar{\psi}_1 + \bar{\omega}_1 = 0, \quad -1 < y < 1 \tag{2.70}$$

$$\bar{\psi}_1'(-1) = 0, \quad \bar{\psi}_1'(+1) = 0, \tag{2.71}$$

and

$$\bar{\omega}_2'' - \sigma\bar{\omega}_2 = f, \quad -1 < y < 1 \tag{2.72}$$

$$\bar{\omega}_2(-1) = 0, \quad \bar{\psi}_2(1) = 1, \tag{2.73}$$

$$\bar{\psi}_2'' - k^2\bar{\psi}_2 + \bar{\omega}_2 = 0, \quad -1 < y < 1 \tag{2.74}$$

$$\bar{\psi}_2'(-1) = 0, \quad \bar{\psi}_2'(+1) = 0, \tag{2.75}$$

The constants ξ_1 and ξ_2 are determined by the algebraic system:

$$\bar{\psi}_1'(-1)\xi_1 + \bar{\psi}_2'(-1)\xi_2 = h_- - \tilde{\psi}'(-1) \tag{2.76}$$

$$\bar{\psi}_1'(1)\xi_1 + \bar{\psi}_2'(1)\xi_2 = h_+ - \tilde{\psi}'(1) \tag{2.77}$$

which can be written as:

$$\mathcal{M}\Xi = \tilde{E}, \tag{2.78}$$

where $\Xi = (\xi_1, \xi_2)^T$ and the matrix $\mathcal{M}$ is the influence matrix.

CHAPTER III

RESULTS

A. Particle Tracking

The study of the Lagrangian characteristics of a flow field enables us to quantify it's mixing properties. Once the velocity field in the domain $\underline{u}(\underline{x}, t)$ has been completely determined by solving for the governing dynamic equations of motion alongwith the boundary conditions, we can solve for the motion of particles in the flow field that are passively advected. We assume that the velocity of the particle is exactly equal to the velocity of the flow filed at its current position, that is the particle instantaneously adjusts its velocity to the ambient flow [8]. Hence, we have:

$$\underline{u}_{particle} = \underline{u}_{fluid} \tag{3.1}$$

We know that the velocity of a particle, $\underline{u}_{particle}$ is given by the rate of change of its position

$$\underline{u}_{particle} = (\frac{dx}{dt}, \frac{dy}{dt}), \tag{3.2}$$

where (x, y) is the position vector of the particle. From (3.1) and (3.2), we get the following system of ordinary differential equations called the advection equation [8]

$$\begin{aligned} \frac{dx}{dt} &= u(x, y, t) \\ \frac{dy}{dt} &= v(x, y, t) \end{aligned} \tag{3.3}$$

We can thereby track the trajectories of advected scalars using (3.3).

As already stated, the flows characterised by chaotic advection have particle trajectories that are extremely sensitive to initial position. Even small errors get magnified exponentially with time. It is therefore important to be able to determine

the velocity vector $\underline{u}$ accurately, at any point in the flow field as a function of time. Hence, the use of low order interpolation schemes such as the 4th order biquadratic algorithm presented in [18] and [19] can lead to low spatial accuracy for passive particle tracking. In the present study, since any field variable is represented by (2.8), its value can be calculated at any position in the flow field with spectral accuracy. The 4th order Runge Kutta method is used to integrate (3.3) numerically with respect to time, and thereby tack the trajectories of the passively advected particles.

B. Poincaré Sections

The numerical construction of Poincaré sections is a standard analysis technique from the theory of dynamical systems. As stated in [8], a Poincaré section may be thought as being generated by stroboscopic illumination of advected points with the period for illumination taken to be the period of one cycle. As given in [20], the poincaré mapping is defined by

$$P_{n+1} = \Psi_p(P_n), \tag{3.4}$$

where Ψ_p is the Poincaré mapping, P_n is the position of the particle at the nth period. Since the flow is periodic with respect to time, the the Poincaré sections are drawn as a function of x and y. The curves that separate the chaotic sea from the quasi-periodic regions are called the Kolmogorov-Arnold-Moser (KAM) curves [9]. The area bounded by the KAM curves and immediately outside them represents the unmixed or poorly mixed regions in the flow field, which are called islands [9]. Islands inhibit good mixing.

In Figure 4, the Poincaré sections for different Reynolds numbers are shown. The maps were obtained by advecting 8 passive tracer particles, initially placed at $(\frac{2}{3}, 0), (\frac{4}{3}, 0), (2, 0), (\frac{8}{3}, 0), (\frac{10}{3}, 0), (4, 0), (\frac{14}{3}, 0), (\frac{16}{3}, 0)$, for 4000 cycles. As can be seen

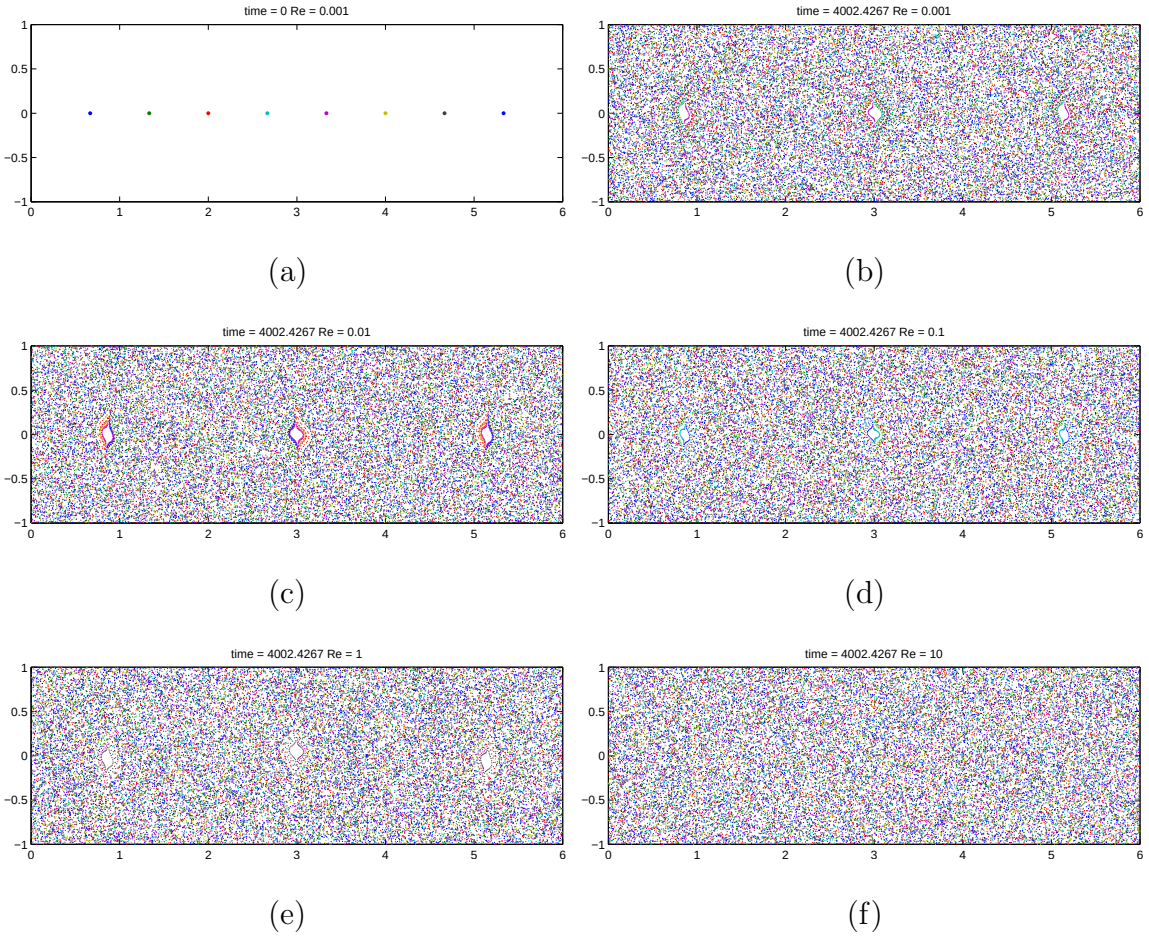


Fig. 4. Poincaré maps for 4000 cycles, for (b) Re = 0.001, (c) Re = 0.01 (d) Re = 0.1, (e) Re = 1 and (f) Re = 10. Note the 3 islands in the flow, surrounded by the chaotic sea, which correspond to the 3 ghost-rods. Also, the islands are shifted from the x axis for Re = 1, and for Re = 10, the entire domain shows chaotic behavior. Figure (a) shows the initial positions of the 8 points used to generate the Poicaré maps.

in Figure 4, there exist 3 islands which correspond to the ghost rods that cause the stirring for $Re \leq 1$. These islands are centred on the x - axis for $Re < 1$ are slightly off-centre for $Re = 1$. For $Re = 10$ however, the entire flow domain is chaotic and there don't exist any quasi-periodic orbits as can be witnessed from the absence of any islands.

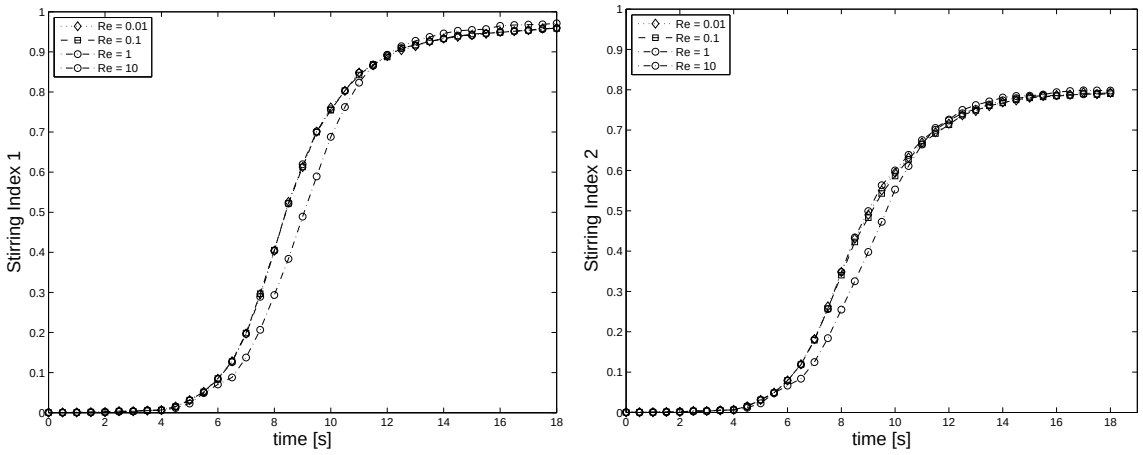
C. Box Counting Method

One way of measuring mixing efficiency is quantifying the dispersion or stirring efficiency for the flow. The box counting method used in [21] by Liu et al. is used to quantify the rate at which dispersion occurs, by counting the number of particles in small uniform boxes. In this method, for a unit-square domain, the box size (s) is given by the relation

$$s \approx 2N^{-1/2}, \tag{3.5}$$

where N is the total number of particles. The above relation is chosen to ensure that for a perfectly random distribution of particles, 98% of the boxes would contain at least one particle [21]. In the current study, the flow domain was divided into 10,000 equally sized boxes, having the same aspect ratio as the half domain from $0 \leq x/h \leq 6$. A total of 40,000 particles was initially put in one box.

Two rules given in [20] are used to calculate the stirring index ϵ. To calculate ϵ using the first rule, the number of filled boxes is divided by the total number of boxes. To calculate ϵ using the second rule, a weighting factor is used. A homogenous mixing state for 40,000 particles distributed amongst 10,000 boxes would be when each box has 4 particles, i.e. $n_{max} = 4$. The weighting factor ω_i for the ith box is given by the relation



(a) (b)

Fig. 5. Comparison of dispersion of passively advected particles using (a) stirring index 1 and (b) stirring index 2 for the box counting method for Re = 0.01, Re = 0.1, Re = 1 and Re = 10

$$\omega_i = \frac{n_i}{n_{max}}, \qquad \text{if } n_i < n_{max}$$
$$\omega_i = 1, \qquad \text{if } n_i \geq n_{max} \tag{3.6}$$

For both rules, the stirring index can be calculated using the relation

$$\epsilon = \frac{1}{K}\sum_{i=1}^{K} \omega_i, \tag{3.7}$$

where K is the total number of boxes and n_{max} is set to 1 for the first rule.

Figure 5 compares the rate of change of stirring index calculated by both rules for Re = 0.01, Re = 0.1, Re = 1 and Re = 10. Figure 6 shows how the particles get dispersed in a chaotic fashion with respect to time.

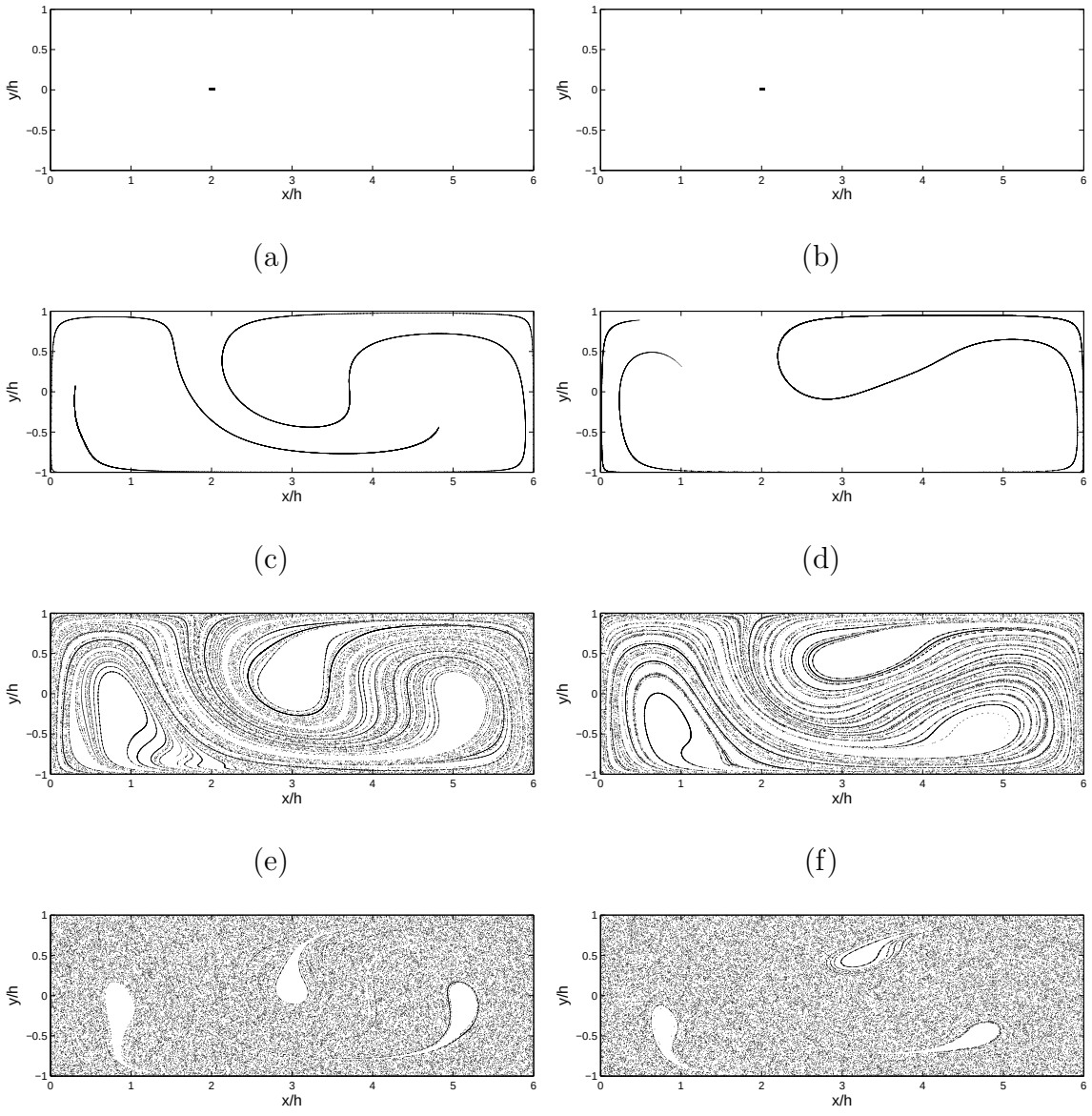


Fig. 6. Comparison of dispersion of passively advected particles for Re = 0.1 (left) and Re = 10 (right), at (a, b) 0, (c, d) 6, (e, f) 10 and (g, h) 15 advection cycles respectively. Note how the islands corresponding to the periodic points for Re = 10 are off-center and smaller in size as compared with Re = 0.1

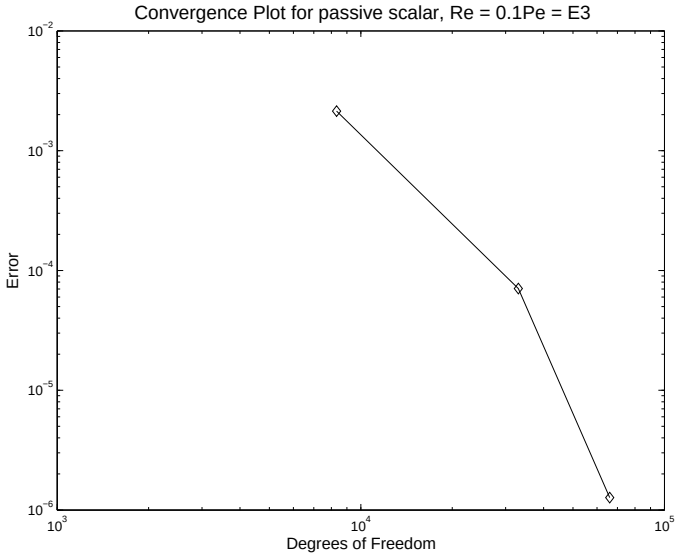
D. Passive Scalar Transport

The aforementioned methods track trajectories of passively advected particles in the flow and thereby can be used to quantify mixing as a function of advection only. To take into the account the effects of diffusion as well, the following passive scalar transport studies are conducted. The transport of passive scalars is simulated using (2.35), once the velocity field has been determined by solving the Navier Stokes equations (2.34) along with the applied boundary conditions. The rate of diffusion in these flows are parameterised by the Peclet Number ($Pe = ReSc$). The mixing characteristics are studied for $Pe = 100$, $Pe = 1000$ and, $Pe = 10,000$, for various Re. Figure 7 shows the convergence for the passive scalar transport simulations for various Peclet numbers. Simulations for higher Peclet numbers were not conducted, since due to limited diffusion, the exponential stretching and folding of the flow leads to sharp concentration gradients at the interface of the fluids, which require high spatial resolutions, which is beyond the scope of the computing power and time available for this study.

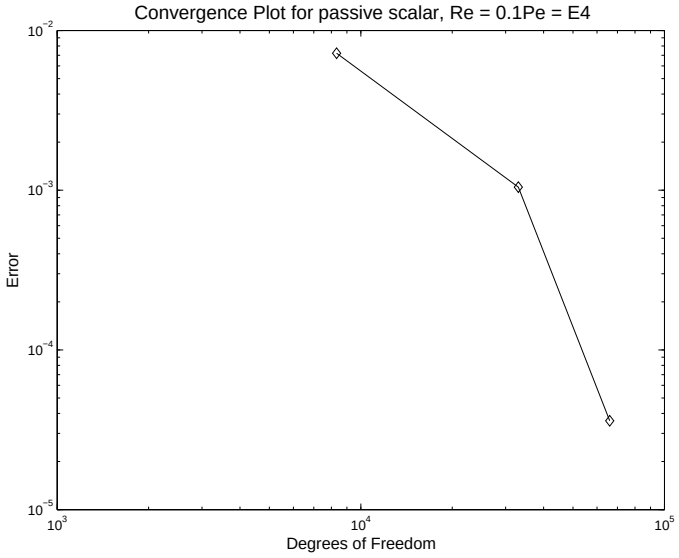
For the simulations, a single passive scalar was used for each run, with $\theta(x, y, t_0) = 0.5$ for $y < 0$ and $\theta(x, y, t_0) = 0.0$ for $y > 0$. Since infinite gradients can't be numerically resolved, a gauss function was used for smoothing of the gradient as shown in Figure 8. From initial conditions, when the passive scalar is completely mixed, the value of its concentration should be 0.25 everywhere in the flow domain.

To quantify the mixing results, the values of θ obtained for each run were calculated at 200 x 1200 evenly spaced grid points at every half cycle. The average mixing at a given time was calculated using the formula

$$M = \frac{1}{N} \sum_{i=1}^{N} \frac{\theta_0 - |\theta_i - \theta_0|}{\theta_0}, \tag{3.8}$$



(a)



(b)

Fig. 7. Convergence plot for passive scalar transport simulations for (a) Pe = 1,000 and (b) Pe = 10,000

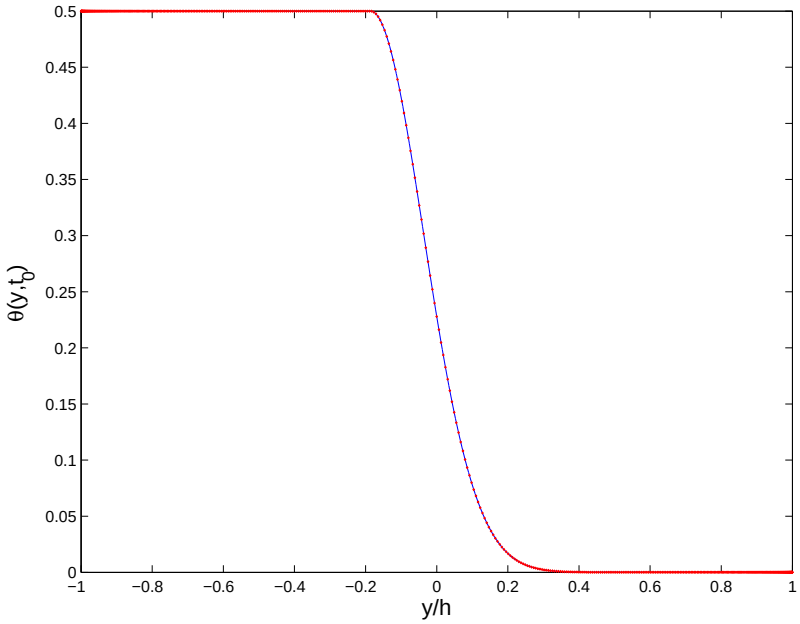
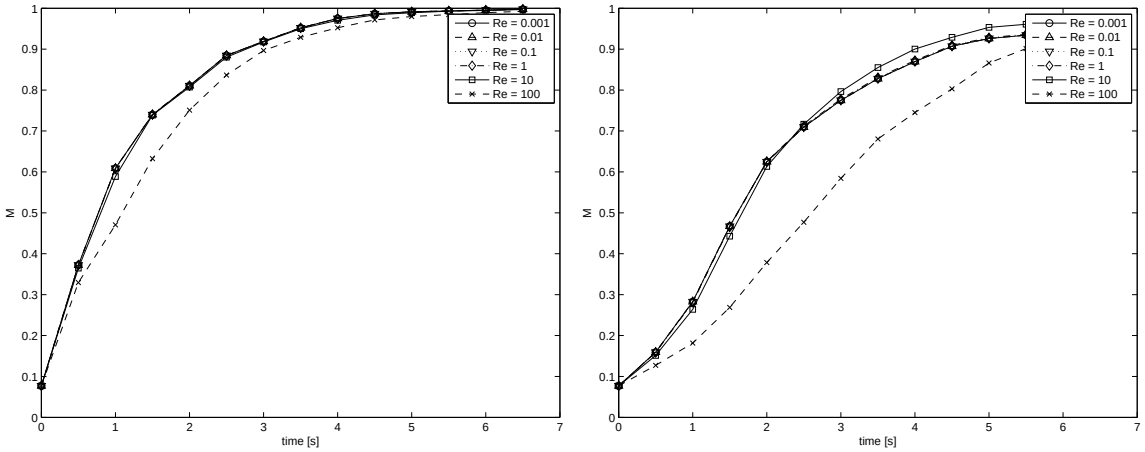


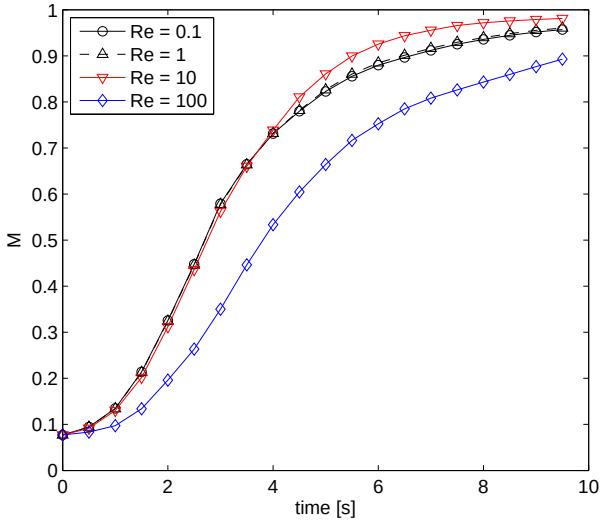
Fig. 8. Initial concentration for passive scalar transport simulations

where N here is the number of grid points, and in the current case from initial conditions, $\theta_0 = 2.5$. A value of $M = 0$ implies a fully unmixed flow field, and a value of $M = 1$ implies a homogenously mixed flow field.

Figure 9 compares the mixing of passive scalars for various Reynolds numbers for a given Peclet number. For a Peclet number of 100, the mixing is driven mainly by the high diffusivity of the passive scalar, and not by the topological chaos of the stirring protocol. This is evident from the high mixing rate at $t = 0$ for $Pe = 100$ for all the Reynolds number cases studied. For a Peclet number of 1,000 and 10,000, it is seen that the mixing rate increases in an exponential manner at first and after a few advection cycles, begins to asymptote to a value of 1.



(a) (b)



(c)

Fig. 9. M as a function of time for (a) Pe = 100, (b) Pe = 1000 and (c) Pe = 10,000

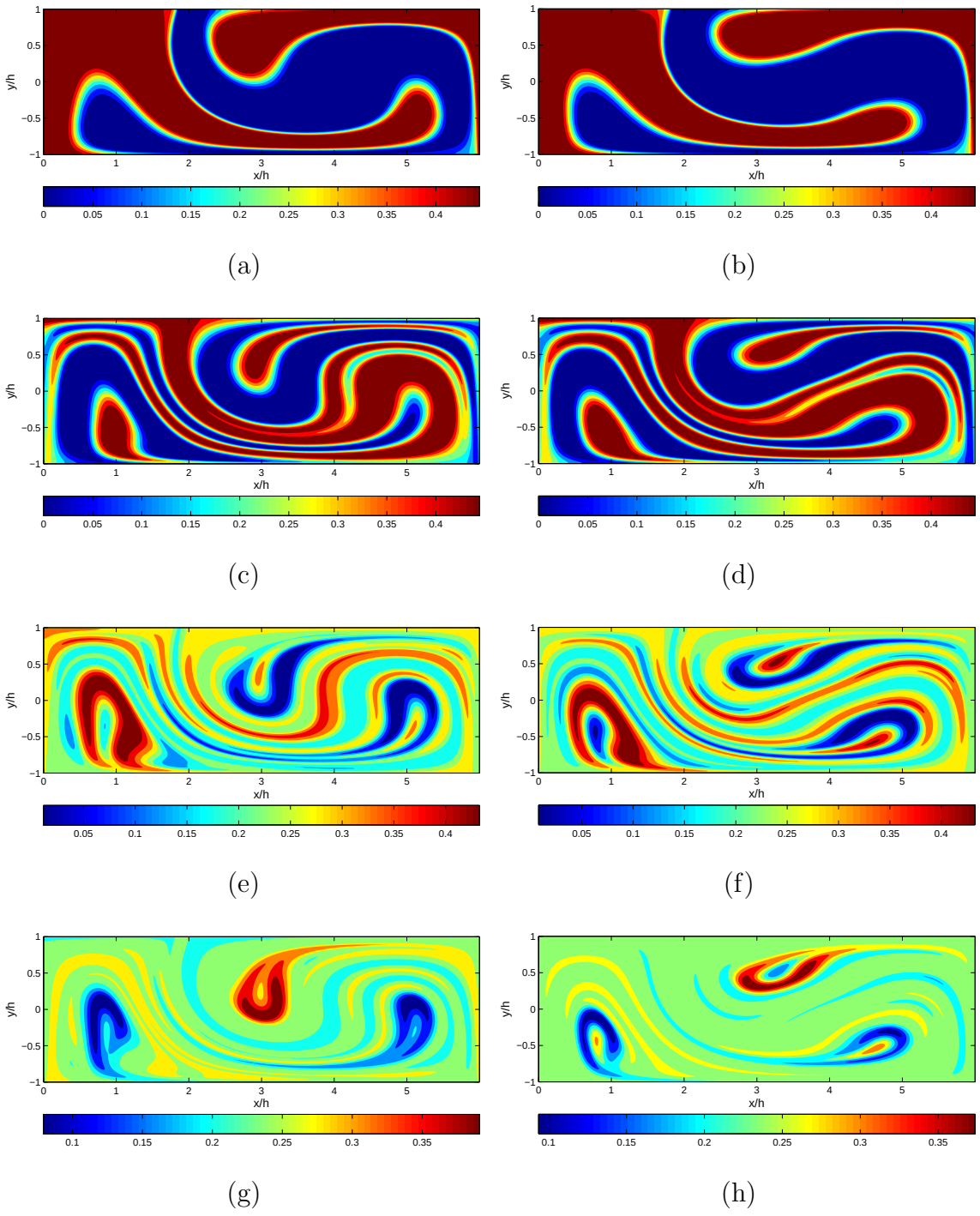


Fig. 10. Contour maps for θ, for Re = 0.1, Pe = 10,000, at (a) 1, (c) 2, (e) 4 and (g) 6 advection cycles and Re = 10, Pe = 10,000 at (b) 1, (d) 2, (f) 4 and (h) 6 advection cycles respectively

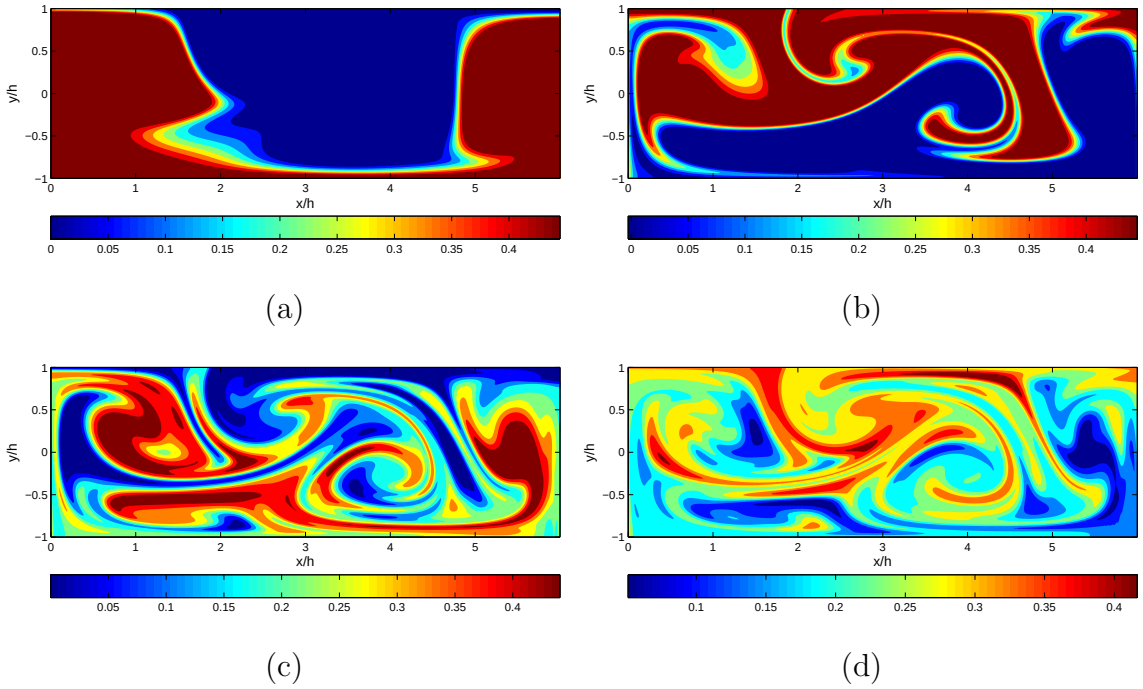


Fig. 11. Contour maps for θ, for Re = 100, Pe = 10,000, at (a) 1, (b) 2, (c) 4 and (d) 6 advection cycles

Figure 10 shows the contour plots of concentration of the passive scalar for Re = 0.1 and Re = 10 for a Peclet number of 10,000 at different times. The three regions of poor mixing in the flow field correspond to the ghost rods that bring about the braiding motion. As evident from the plots, the size of ghost rods for Re = 10 is smaller than that for Re = 0.1. Figure 11 shows the development of the passive scalar field for Re = 100 and Pe = 10,000. Although the flow pattern is more complex than the lower Reynolds number cases, the mixing achieved is much less efficient. Figure 12 gives an idea of the braiding pattern acieved for various Reynolds numbers.

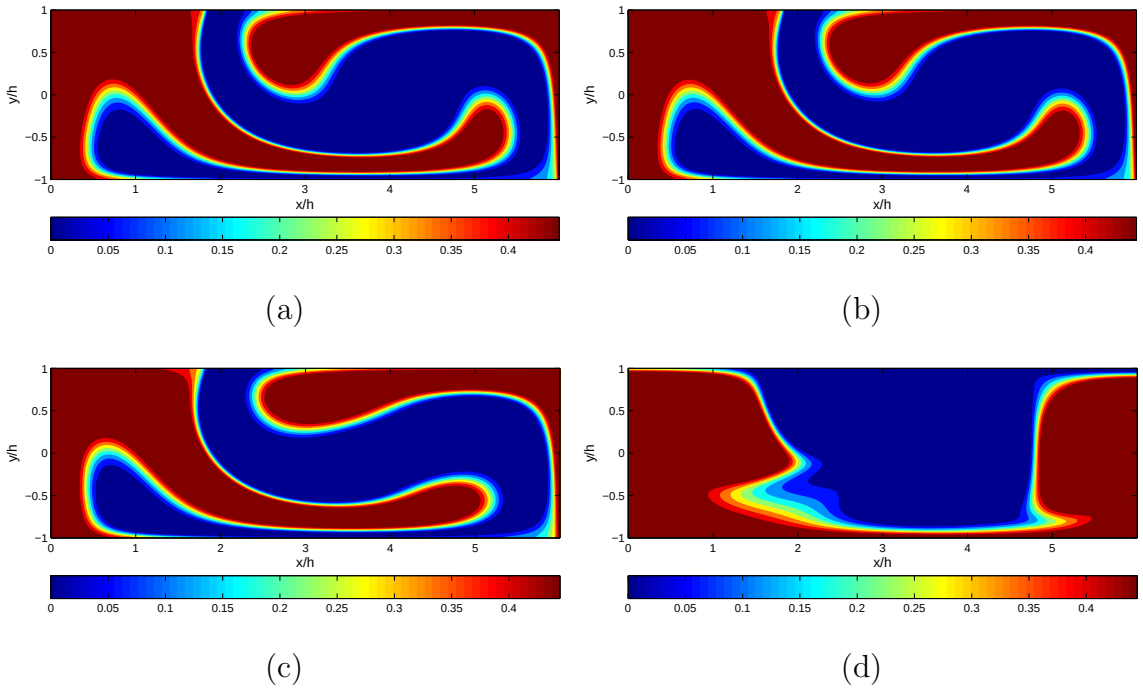


Fig. 12. The above contour maps for θ, for Pe = 10,000 and (a) Re = 0.1, (b) Re = 1, (c) Re = 10, (d) Re = 100 respectively, at 1 advection cycle show how inertia affects the braiding action of the ghost rods

CHAPTER IV

CONCLUSIONS

The mixing characteristics and topological choas of chaotic advection using mixing by "ghost-rods" was studied, using standard techniques such as Poincaré sections, the box counting method and passive scalar transport. A Fourier-Chebyshev spectral method was used to solve the Navier-Stokes and scalar transport equations to get spectrally accurate values of flow variables.

It was observed for $Re \leq 1$, that topological chaos and mixing efficiency are independant of Reynolds number. From this it can be concluded that the Stokes flow assumptions work well for $Re \leq 1$, and inertial effects can be considered to be negligible in this regime.

For $Re = 10$, it was observed in the case of the box counting method as well as the passive scalar transport, that the rate of mixing was slightly lower than for flows with $Re \leq 1$ in the region of exponential growth. This can be attributed to the fact that inertia alters the way the periodic points are switched in every braiding cycle, which leads to a reduction in the stretching and folding caused by the braiding. However, inertial effects also lead to the destruction of non-chaotic islands as seen in the Poincaré map for $Re = 10$. Due to this, at later times, we get better mixing than flows at lower Reynolds numbers.

For $Re = 100$, because of the sluggishness created in the flow due to inertia, no braiding occurs. Hence for $Re = 100$ we get the slowest rates of mixing. To get better mixing efficiency at such high Reynolds numbers, a different protocol (different boundary conditions, switching times, etc.) will need to be adopted.

REFERENCES

[1] D. A. Zumbrunnen and S. Inamdar, "Novel sub-micron highly multi-layered polymer films formed by continuous flow chaotic mixing," *Chemical Engineering Science*, vol. 56, pp. 3893–3897, 2001.

[2] H. Aref, "Stirring by chaotic advection," *J. Fluid Mech*, vol. 143, pp. 1–21, 1984.

[3] S. W. Jones, O. M. Thomas, and H. Aref, "Chaotic advection by laminar flow in a twisted pipe," *J. Fluid Mech.*, vol. 209, pp. 335–357, 1989.

[4] H. Aref, S. W. Jones, S. Mofina, and I. Zawadzki, "Vortices, kinematics and chaos," *Physica D*, vol. 37, pp. 423–440, 1989.

[5] H. Aref and S. W. Jones, "Enhanced separation of diffusing particles by chaotic advection," *Phys. Fluids A*, vol. 1, pp. 470–474, March 1988.

[6] P. L. Boyland, H. Aref, and M. A. Stremler, "Topological fluid mechanics of stirring," *J. Fluid Mech.*, vol. 403, pp. 277–304, 2000.

[7] P. Boyland, M. Stremler, and H. Aref, "Topological fluid mechanics of point vortex motions," *Physica D*, vol. 175, pp. 69–95, 2003.

[8] H. Aref, "The development of chaotic advection," *Physics of Fluids*, vol. 14, pp. 1315–1325, 2002.

[9] S. Wiggins and J. M. Ottino, "Foundations of chaotic mixing," *Phil. Trans. R. Soc. Lond. A*, vol. 362, pp. 937–970, March 2004.

[10] P. Dutta and R. Chevray, "Inertial effects in chaotic mixing with diffusion," *J. Fluid Mech.*, vol. 285, pp. 1–16, 1995.

[11] D. M. Hobbs and F. J. Muzzio, "Reynolds number effects on laminar mixing in the kenics static mixer," *Chemical Engineering Journal*, vol. 70, pp. 93–104, 1998.

[12] M. J. Clifford, S. M. Cox, and M. D. Finn, "Reynolds number effects in a simple planetary mixer," *Chemical Engineering Science*, vol. 59, pp. 3371–3379, 2004.

[13] J. Wang, L. Feng, J. M. Ottino, and R. Lueptow, "Inertial effects on chaotic advection and mixing in a 2d cavity," *Ind. Eng. Chem. Res.*, vol. 48, pp. 2436–2442, 2009.

[14] M. A. Stremler and J. Chen, "Generating topological chaos in lid-driven cavity flow," *Physics of Fluids*, vol. 19, p. 103602, October 2007.

[15] C. Canuto, M. Y. Hussaini, A. Quarteroni, and J. Thomas A. Zang, *Spectral Methods, Fundamentals in Single Domains*. Berlin: Springer-Verlag, 2006.

[16] M. A. Stremler, private communications, VirginiaTech, Blacksburg, February 2009.

[17] R. Peyret, *Spectral Methods for Incompressible Viscous Flow*. New York: Springer-Verlag Inc, 2002.

[18] G. Coppola, S. J. Sherwin, and J. Peiro, "Nonlinear particle tracking for high-order elements," *J. Comp. Phys.*, vol. 172, pp. 356–386, 2001.

[19] A. Souvaliotis, S. C. Jana, and J. M. Ottino, "Potentialities and limitations of mixing simulations," *AIChE J.*, vol. 41, pp. 1605–1621, 1995.

[20] H. J. Kim, "Theoretical and numerical studies of chaotic mixing," Ph.D. dissertation, Texas A&M University, College Station, Texas, 2008.

[21] M. Liu, F. J. Muzzio, and R. L. Peskin, "Quantification of mixing in aperiodic chaotic flows," *Chaos, Solitons & Fractals*, vol. 4, pp. 869–893, 1994.

APPENDIX A

SOURCE CODE

Main Function

```
#include<fftw3.h>
#include<stdio.h>
#include<stdlib.h>
#include<stddef.h>
#include<ctype.h>
#include<math.h>
#include<string.h>
#include<mpi.h>
#include<time.h>
#define STRMAX 50
#define PI 3.141592653589793
#define ITER 600000000
#define ERROR 1e-14
#define ERROR2 1e-14

//#############################################################################
//#define PAS_SCAL
//#define POINCARE
#define TRACKER
//#define DEBUG
//#############################################################################


int main(argc, argv)
int argc;
char *argv[];
{
// Declare Variables ============================================================
 char outputfile[5*STRMAX],line[STRMAX],temp[STRMAX],inputfile[STRMAX*5],input1[
     STRMAX*2],outstr[STRMAX*5], outstr1[STRMAX*5], outstr2[5*STRMAX],psinitfile[5*
     STRMAX], valoutput[STRMAX], inputfpcare[STRMAX*5], description[STRMAX*5];
 int SIZE, xres, yres,xresps,yresps,xresdeal, yresdeal,xresdealps,yresdealps,
     switchcount = 1,copycount = 0,copycount2 = 0, reversecount = 0, eventflag = 0,
     theta2flag = 1;
 int i,j,k,rk4count=3,hscount = 0,xn,yn,coordx,coordy,phase=0,hsratio,rk4ratio;//
     outcount = 0;
 int zerovelflag = 0;
 double Re, deltaTBE, deltaT2BE, deltaTps,deltaTuv, hs, U, xlen, ylen, *boundaryval,
     *boundaryval1, *boundaryval2, *boundaryval0, *boundaryvalt, *coord, *u, *
     boundaryvalop, **uRK4, **uRK4i, **vRK4, **vRK4i, *temppoint, *coordpcare, *
     coordnpcare, *coordfinalpcare, coordouttimepcare;
 double Lcheb = 2.0;
 double Lfour, time,boundtime,outtime,outtimeps,switchtime, reversetime ,outputcount
     =0.0,outputsperfile = 1.0,outputcountps=0.0, coordoutcountpcare = 0.0,
     coordfinalcountpcare = 0.0;
 double flag = 0.0;
 char option,c;
 char output1[STRMAX*5],outputoption[STRMAX];
 char output2[] = "./output/output2.txt";
 time_t start,end;
 double dif;
 long unsigned int iter,itconv,iterations, iterationsps,coordn, coordnumpcare;
```

```
 //Variables for MPI==========theta
 int numprocs, rank, dest, source, rc, count, tag,numtasks,*tasks,*tasksnum;
 int n1,n2,N,rem;

 int numtasksps1,*tasksps1,*tasksnumps1, numtasksps2,*tasksps2,*tasksnumps2;
 int n1ps1,n2ps1,Nps1,remps1, n1ps2,n2ps2,Nps2,remps2;
 //============================

int *procarray;
int procalloc_theta[2], procallocnum_theta[2];
int rank_theta1 = -1, rank_theta2 = -1;

 MPI_Group worldgroup, group_theta1, group_theta2;
 MPI_Comm comm_theta1, comm_theta2;
// MPI_Status Stat;

 FILE *outputfp, *outputTfp;
 FILE *output2fp;
 FILE *outputmaxfp;
 FILE *inputfp;

 fftw_complex *omegak, *omegaktilde, *omegak1, *omegak2, *omegakn, *omegaknmin1, *
     omegaknmin11,*omegaknmin2, *theta1k, *theta2k, *theta1kn, *theta2kn, *
     theta1knmin1, *theta2knmin1, *theta1knmin11, *theta2knmin11, *theta1knmin2, *
     theta2knmin2, *theta1k2nmin1, *theta1k2nmin2,*theta2k2nmin1, *theta2k2nmin2;
 fftw_complex *psik, *psiktilde, *psik1, *psik2, *psikn, *psiknmin1, *psiknmin11,*
     psiknmin2, *psipskn, *psipsknmin1, *psipsknmin11, *psipsknmin2;
 fftw_complex *in, *out, *inps, *outps,*indeal, *outdeal, *indealps, *outdealps;;
 fftw_complex *incheb,*outcheb, *inchebps,*outchebps, *inchebdeal,*outchebdeal, *
     inchebdealps,*outchebdealps;
 fftw_complex *Ainv, *lambda;
 fftw_complex *temp1, *temp2,*temp2D1,*temp2D2, *temp1ps, *temp2ps;
 fftw_complex *d2psidx2k, *d2psidxdyk, *d2psidy2k,*dpsidxk,*dpsidyk;
 fftw_complex *uk, *vk, *ukn, *vkn, *ukint, *vkint;
 fftw_complex *K;
 double x[2][2],F[2],J[4],detJ,Jinv[4],utemp;
 fftw_plan pfor, pback, pchebfor,pchebback,pcheb, pforps, pbackps, pchebforps,
     pchebbackps,pchebps, pfordeal, pbackdeal, pchebfordeal,pchebbackdeal,pchebdeal,
     pfordealps, pbackdealps, pchebfordealps,pchebbackdealps,pchebdealps;

 fftw_complex *NLn, *NLnmin1, *omegan, *omeganmin1;

 double Sc1 = 999.5, Sc2 = 1000.5;
// ========================================================================

start = clock();
// MPI stuff ==============================================================
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 int blockcounts[1];
 MPI_Status Stat;
 MPI_Datatype MPI_complex,oldtypes[1];
 MPI_Aint offsets[1];
 offsets[0] = 0;
 blockcounts[0] = 2;
 oldtypes[0] = MPI_DOUBLE;

 MPI_Type_struct(1,blockcounts,offsets,oldtypes,&MPI_complex);
 MPI_Type_commit (&MPI_complex);
// ========================================================================

// File open ==============================================================
```

```
 if(rank == 0)
         {output2fp = fopen(output2,"w");
         strcpy(input1,"./input/sp2Dinp.txt");
                 inputfp = fopen(input1,"r");
                 if(inputfp == NULL)
                        {printf("\nFile not found!!!\n"); return 0;}
        }
// ============================================================================

// Input Data ================================================================
if(rank == 0)
         {
          fnamesearch("DESCRIPTION","=", inputfp);
          fgetterm(inputfp,description,STRMAX*5);

          fnamesearch("INPUTFILE","=", inputfp);
          fgetterm(inputfp,inputfile,STRMAX*5);

    if(readinput(inputfile, &xres, &yres,&xlen,&ylen,&xn,&yn,&U, &boundaryval)); else
          {printf("\nFile not found!!!!\n");return 1;}

     normalisebound(boundaryval, U, xres-1);

//       printf("\nboundaryval:\n");print2D(boundaryval,2,xres);
          fnamesearch("OUTPUTDIR","=", inputfp);
          fgetterm(inputfp,output1,STRMAX*5);

#ifdef PAS_SCAL
          fnamesearch("PSINPUTFILE","=", inputfp);
          fgetterm(inputfp,psinitfile,STRMAX*5);
#endif
#ifdef POINCARE
          fnamesearch("INPUTFILEPCARE","=", inputfp);
          fgetterm(inputfp,inputfpcare,STRMAX*5);
#endif

#ifdef TRACKER
          fnamesearch("INPUTFILETRACKER","=", inputfp);
          fgetterm(inputfp,inputfpcare,STRMAX*5);
#endif


          fnamesearch("Re","=", inputfp);
          fgetterm(inputfp,line,STRMAX);

          strcpy(temp,line);
          Re = atof(temp);
          strcpy(outputfile,output1);
          strcat(outputfile,"maxRe");
          strcat(output1,"Re");
          for(i = 0;i<strlen(line);i++)
                 if(line[i] == '.')
                         line[i] ='_';
          strcat(outputfile, line);
          strcat(output1, line);
          fnamesearch("TIME","=", inputfp);
          fgetterm(inputfp,line,STRMAX);

          strcpy(temp,line);
          time = atof(temp);
//        time = time/(ylen/U);
          strcat(outputfile, "T");
          strcat(output1, "T");
```

```
         for(i = 0;i<strlen(line);i++)
                if(line[i] == '.')
                        line[i] ='_';

         strcat(outputfile, line);
         strcat(output1, line);

#ifdef PAS_SCAL
         fnamesearch("DELTATPS","=", inputfp);
         fgetterm(inputfp,line,STRMAX);
         utemp = atof(line);
        deltaTps = utemp;

         fnamesearch("HS","=", inputfp);
         fgetterm(inputfp,line,STRMAX);
         utemp = atof(line);
         hs = utemp;
#endif
         fnamesearch("DELTAT2BE","=", inputfp);
         fgetterm(inputfp,line,STRMAX);
         utemp = atof(line);
         deltaT2BE = utemp;
#ifdef PAS_SCAL
         fnamesearch("DELTATUV","=", inputfp);
         fgetterm(inputfp,line,STRMAX);
         utemp = atof(line);
         deltaTuv = utemp;
#endif

        deltaTBE = deltaT2BE/100;
        rk4ratio = (int)(deltaTps/hs);
        hsratio = (int)(hs/deltaT2BE/2.0);

         fnamesearch("OUTPERIOD","=", inputfp);
         fgetterm(inputfp,line,STRMAX);
         outtime = atof(line);
//       outtime = outtime/(ylen/U);
#ifdef PAS_SCAL
         fnamesearch("OUTPERIODPS","=", inputfp);
         fgetterm(inputfp,line,STRMAX);
         outtimeps = atof(line);
//       outtimeps = 0.1;
//       outtimeps = outtimeps/(ylen/U);
#endif

#ifdef POINCARE
         fnamesearch("OUTPERIODPCARE","=", inputfp);
         fgetterm(inputfp,line,STRMAX);
         coordouttimepcare = atof(line);
printf("\n OUTPERIODPCARE = %g\n",coordouttimepcare);
//       outtimeps = 0.1;
//       outtimeps = outtimeps/(ylen/U);
#endif

#ifdef TRACKER
         fnamesearch("OUTPERIODTRACKER","=", inputfp);
         fgetterm(inputfp,line,STRMAX);
         coordouttimepcare = atof(line);
printf("\n OUTPERIODTRACKER = %g\n",coordouttimepcare);
//       outtimeps = 0.1;
//       outtimeps = outtimeps/(ylen/U);
#endif
```

```
        fnamesearch("OUTPUTOPTION","=", inputfp);
        fgetterm(inputfp,outputoption,STRMAX);

        fnamesearch("VALOUTPUT","=", inputfp);
        fgetterm(inputfp,valoutput,STRMAX);

        fnamesearch("OUTPUTSPERFILE","=", inputfp);
        fgetterm(inputfp,line,STRMAX);
        outputsperfile = atof(line);

        strcat(output1, "P");

        fnamesearch("BOUNDPERIOD","=", inputfp);
        fgetterm(inputfp,line,STRMAX);
        strcpy(temp,line);

        boundtime = atof(temp);
//      boundtime = boundtime/(ylen/U);

        for(i = 0;i<strlen(line);i++)
               if(line[i] == '.')
                       line[i] ='_';

        strcat(output1, line);

        strcat(output1, "S");
        fnamesearch("SWITCHPERIOD","=", inputfp);
        fgetterm(inputfp,line,STRMAX);
        strcpy(temp,line);

        switchtime = atof(temp);
//      switchtime = switchtime/(ylen/U);

        for(i = 0;i<strlen(line);i++)
               if(line[i] == '.')
                       line[i] ='_';

        strcat(output1, line);

        strcat(output1, "R");
        fnamesearch("REVERSEPERIOD","=", inputfp);
        fgetterm(inputfp,line,STRMAX);
        strcpy(temp,line);

        reversetime = atof(temp);
//      reversetime = reversetime/(ylen/U);

        for(i = 0;i<strlen(line);i++)
               if(line[i] == '.')
                       line[i] ='_';

        strcat(output1, line);
        strcpy(outstr,output1);

        strcat(output1,".txt");

        printf("\n output1: %s\n",output1);
#ifdef PAS_SCAL
        fnamesearch("PSRES",":", inputfp);
        fgetterm(inputfp,line,STRMAX);
        xresps = atoi(line);
        fgetterm(inputfp,line,STRMAX);
        yresps = atoi(line);
```

```
         fnamesearch("SC1","=", inputfp);
         fgetterm(inputfp,line,STRMAX);
         Sc1 = atof(line);

         fnamesearch("SC2","=", inputfp);
         fgetterm(inputfp,line,STRMAX);
         Sc2 = atof(line);
#endif
         fnamesearch("COORDX","=", inputfp);
         fgetterm(inputfp,line,STRMAX);
         coordx = atoi(line);

         fnamesearch("COORDY","=", inputfp);
         fgetterm(inputfp,line,STRMAX);
         coordy = atoi(line);
         coordn = coordx*coordy;

        xresdeal = (xres/2)*3;
        yresdeal = yres*2;
#ifdef PAS_SCAL
        xresdealps = (xresps/2)*3;
        yresdealps = yresps*2;

         fnamesearch("THETA2",":", inputfp);
         fgetterm(inputfp,line,STRMAX);

        if(!strcmp(line,"NO"))
                {theta2flag = 0; printf("\nTHETA2 FLAG = 0!!!!!!!!!!!!!!!!!!!!!\n\n
                    ");}
        else theta2flag = 1;

         fnamesearch("ZEROVEL",":", inputfp);
         fgetterm(inputfp,line,STRMAX);

        if(!strcmp(line,"YES"))
                {zerovelflag = 1; printf("\nZEROVEL FLAG = 1!!!!!!!!!!!!!!!!!!!!\n\
                    n");}
        //else theta2flag = 1;
        if (zerovelflag == 1)
                {for (i=0;i<2*xres;i++)
                                boundaryval[i] = 0.0;
                }
printf("\nxresps = %d, yresps = %d\n",xresps,yresps);
#endif
        printf("\nRe = %g, time = %g, deltaT2BE = %g deltaTBE = %g, boundtime = %g,
            switchtime = %g, reversetime = %g, outtime = %g, Sc1 = %g, Sc2 = %g,
            outtimeps = %g\n",Re, time, deltaT2BE, deltaTBE, boundtime,switchtime,
            reversetime, outtime, Sc1, Sc2,outtimeps);
        printf("\nrk4ratio = %d, hsratio = %d", rk4ratio,hsratio);
        Lfour = xlen/ylen;//*((xres)*1.0)/(xres*1.0);
        printf("\nLfour = %g\n",Lfour);

        }
// ============================================================================

if(rank == 0) printf("\nBcast Started\n");

 MPI_Bcast (&(xres),1,MPI_INT,0,MPI_COMM_WORLD);
 MPI_Bcast (&(yres),1,MPI_INT,0,MPI_COMM_WORLD);
#ifdef PAS_SCAL
 MPI_Bcast (&(xresps),1,MPI_INT,0,MPI_COMM_WORLD);
 MPI_Bcast (&(yresps),1,MPI_INT,0,MPI_COMM_WORLD);
#endif
 MPI_Bcast (&(xresdeal),1,MPI_INT,0,MPI_COMM_WORLD);
```

```
 MPI_Bcast (&(yresdeal),1,MPI_INT,0,MPI_COMM_WORLD);
#ifdef PAS_SCAL
 MPI_Bcast (&(xresdealps),1,MPI_INT,0,MPI_COMM_WORLD);
 MPI_Bcast (&(yresdealps),1,MPI_INT,0,MPI_COMM_WORLD);
#endif
 MPI_Bcast (&(coordx),1,MPI_INT,0,MPI_COMM_WORLD);
 MPI_Bcast (&(coordy),1,MPI_INT,0,MPI_COMM_WORLD);
 MPI_Bcast (&(coordn),1,MPI_UNSIGNED_LONG,0,MPI_COMM_WORLD);
 MPI_Bcast (&(Re),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast (&(time),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast (&(deltaTBE),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast (&(deltaT2BE),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
#ifdef PAS_SCAL
 MPI_Bcast (&(deltaTps),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast (&(deltaTuv),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast (&(rk4ratio),1,MPI_INT,0,MPI_COMM_WORLD);
 MPI_Bcast (&(hsratio),1,MPI_INT,0,MPI_COMM_WORLD);
 MPI_Bcast (&(hs),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
#endif
 MPI_Bcast (&(U),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast (&(xlen),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast (&(ylen),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast (&(Lfour),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast (&(xn),1,MPI_INT,0,MPI_COMM_WORLD);
 MPI_Bcast (&(yn),1,MPI_INT,0,MPI_COMM_WORLD);
 MPI_Bcast (&(boundtime),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast (&(switchtime),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast (&(reversetime),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast (&(outtime),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
#ifdef PAS_SCAL
 MPI_Bcast (&(outtimeps),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
#endif
 MPI_Bcast (outputoption,STRMAX,MPI_CHAR,0,MPI_COMM_WORLD);
 MPI_Bcast (valoutput,STRMAX,MPI_CHAR,0,MPI_COMM_WORLD);
#ifdef PAS_SCAL
 MPI_Bcast (&(Sc1),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast (&(Sc2),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast (&(theta2flag),1,MPI_INT,0,MPI_COMM_WORLD);
#endif

#ifdef POINCARE
 MPI_Bcast (&(coordouttimepcare),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
#endif

#ifdef TRACKER
 MPI_Bcast (&(coordouttimepcare),1,MPI_DOUBLE,0,MPI_COMM_WORLD);
#endif

//printf("\n%d time = %g\n",rank,time);
 MPI_Barrier (MPI_COMM_WORLD);
//printf("\n%d bef bound = %g\n",rank,time);
 if(rank>0)
        boundaryval = malloc(sizeof(double)*2*xres);
 MPI_Bcast (boundaryval,2*xres,MPI_DOUBLE,0,MPI_COMM_WORLD);


//if(rank>0)
//      {printf("\nboundaryval:\n");print2D(boundaryval,2,xres);}
//printf("\n%d aft bound = %g\n",rank,time);
if(rank == 0) printf("\nMem Allocation Started\n");
// Allocate Memory=============================================================
 in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*xres);
 out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*xres);
 incheb = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*2*(yres-1));
```

```
 outcheb = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*2*(yres-1));
#ifdef PAS_SCAL
 inps = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*xresps);
 outps = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*xresps);
 inchebps = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*2*(yresps-1));
 outchebps = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*2*(yresps-1));
#endif

 indeal = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*xresdeal);
 outdeal = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*xresdeal);
 inchebdeal = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*2*(yresdeal-1));
 outchebdeal = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*2*(yresdeal-1));
#ifdef PAS_SCAL
 indealps = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*xresdealps);
 outdealps = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*xresdealps);
 inchebdealps = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*2*(yresdealps-1));
 outchebdealps = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*2*(yresdealps-1));
#endif

 omegak = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 omegaktilde = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 omegak1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 omegak2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 psik = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 psiktilde = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 psik1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 psik2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 temp1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 omegakn = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 omegaknmin1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 omegaknmin11 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 omegaknmin2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 psikn = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 psiknmin1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 psiknmin11 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 psiknmin2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);

#ifdef POINCARE
 uk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 vk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 ukn = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 vkn = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 ukint = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 vkint = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
#endif

#ifdef TRACKER
 uk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 vk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 ukn = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 vkn = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 ukint = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
 vkint = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);
#endif

#ifdef PAS_SCAL
 temp1ps = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);
 temp2ps = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);
 psipskn = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);
 psipsknmin1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);
 psipsknmin11 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);
 psipsknmin2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);
#endif
 Ainv = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*4);
```

```
 lambda = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*2);                    479
 temp2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);                  480
 boundaryvalop = malloc(sizeof(double)*2*xres);                                       481
 boundaryval1 = malloc(sizeof(double)*2*xres);                                        482
 boundaryval2 = malloc(sizeof(double)*2*xres);                                        483
 boundaryval0 = malloc(sizeof(double)*2*xres);                                        484
 boundaryvalt = malloc(sizeof(double)*2*xres);                                        485
 temp2D1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);                486
 temp2D2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);                487
 coord = malloc(sizeof(double)*2*coordn);                                             488
 u = malloc(sizeof(double)*coordn);                                                   489
 d2psidx2k = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);              490
 d2psidxdyk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);             491
 d2psidy2k = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);              492
 dpsidxk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);                493
 dpsidyk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xres*yres);                494
                                                                                      495
 NLn = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresdeal*yresdeal);            496
 NLnmin1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresdeal*yresdeal);        497
 omegan = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresdeal*yresdeal);         498
 omeganmin1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresdeal*yresdeal);     499
                                                                                      500
#ifdef PAS_SCAL                                                                       501
 uRK4 = malloc(sizeof(double*)*(int)(rk4ratio)*2*3+1);                                502
 uRK4i = malloc(sizeof(double*)*(int)(rk4ratio)*2*3+1);                               503
 vRK4 = malloc(sizeof(double*)*(int)(rk4ratio)*2*3+1);                                504
 vRK4i = malloc(sizeof(double*)*(int)(rk4ratio)*2*3+1);                               505
 for(i = 0; i<(int)(rk4ratio)*2*3+1;i++)                                              506
        {uRK4[i] = malloc(sizeof(double)*xres*yres);                                  507
        uRK4i[i] = malloc(sizeof(double)*xres*yres);                                  508
        vRK4[i] = malloc(sizeof(double)*xres*yres);                                   509
        vRK4i[i] = malloc(sizeof(double)*xres*yres);                                  510
        }                                                                             511
 theta1k = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);            512
 theta2k = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);            513
 theta1kn = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);           514
 theta2kn = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);           515
 theta1knmin1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);       516
 theta2knmin1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);       517
 theta1knmin11 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);      518
 theta2knmin11 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);      519
 theta1knmin2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);       520
 theta2knmin2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresps*yresps);       521
 theta1k2nmin1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresdealps*          522
     yresdealps);
 theta1k2nmin2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresdealps*          523
     yresdealps);
 theta2k2nmin1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresdealps*          524
     yresdealps);
 theta2k2nmin2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*xresdealps*          525
     yresdealps);
#endif                                                                                526
 tasks = malloc(sizeof(int)*numprocs);                                                527
 tasksnum = malloc(sizeof(int)*numprocs);                                             528
#ifdef PAS_SCAL                                                                       529
 tasksps1 = malloc(sizeof(int)*numprocs);                                             530
 tasksnumps1 = malloc(sizeof(int)*numprocs);                                          531
                                                                                      532
 tasksps2 = malloc(sizeof(int)*numprocs);                                             533
 tasksnumps2 = malloc(sizeof(int)*numprocs);                                          534
#endif                                                                                535
// ===================================================================================536
 for(i = 0;i<xres*2;i++)                                                              537
        boundaryvalt[i] = boundaryval[i];                                             538
```

```
 if(rank == 0) printf("\nGet n1n2:\n");
getn1n2(&n1, &n2, rank,tasks, tasksnum, &numtasks,numprocs, ceil1(xres,2)+1);
//#ifdef PAS_SCAL
// getn1n2(&n1ps, &n2ps, rank,tasksps, tasksnumps, &numtasksps,numprocs, ceil1(xresps
    ,2)+1);
//#endif

 {
//      printf("\nrank = %d, n1 = %d, n2 = %d, numtasks = %d\n",rank,n1,n2,numtasks);
//      printf("\nrank = %d, n1ps = %d, n2ps = %d, numtasksps = %d\n",rank,n1ps,n2ps,
    numtasksps);
//   if(rank == 0){printf("\ntasks = \t");print2Dint(tasks,1,numtasks);printf("\
    ntasksnum = \t");print2Dint(tasksnum,1,numtasks);}
if(rank == 0) printf("\nFFTW Plans:\n");
// FFTW PLANS=====================================================================
         pfor = fftw_plan_dft_1d(xres, in, out, FFTW_FORWARD, FFTW_EXHAUSTIVE);
         pback = fftw_plan_dft_1d(xres, in, out, FFTW_BACKWARD, FFTW_EXHAUSTIVE);
         pchebfor = fftw_plan_dft_1d(2*(yres-1), incheb, outcheb, FFTW_FORWARD,
             FFTW_EXHAUSTIVE);
         pchebback = fftw_plan_dft_1d(2*(yres-1), incheb, outcheb, FFTW_BACKWARD,
             FFTW_EXHAUSTIVE);
#ifdef PAS_SCAL
         pforps = fftw_plan_dft_1d(xresps, inps, outps, FFTW_FORWARD, FFTW_EXHAUSTIVE
             );
         pbackps = fftw_plan_dft_1d(xresps, inps, outps, FFTW_BACKWARD,
             FFTW_EXHAUSTIVE);
         pchebforps = fftw_plan_dft_1d(2*(yresps-1), inchebps, outchebps,
             FFTW_FORWARD, FFTW_EXHAUSTIVE);
         pchebbackps = fftw_plan_dft_1d(2*(yresps-1), inchebps, outchebps,
             FFTW_BACKWARD, FFTW_EXHAUSTIVE);
#endif

         pfordeal = fftw_plan_dft_1d(xresdeal, indeal, outdeal, FFTW_FORWARD,
             FFTW_EXHAUSTIVE);
         pbackdeal = fftw_plan_dft_1d(xresdeal, indeal, outdeal, FFTW_BACKWARD,
             FFTW_EXHAUSTIVE);
         pchebfordeal = fftw_plan_dft_1d(2*(yresdeal-1), inchebdeal, outchebdeal,
             FFTW_FORWARD, FFTW_EXHAUSTIVE);
         pchebbackdeal = fftw_plan_dft_1d(2*(yresdeal-1), inchebdeal, outchebdeal,
             FFTW_BACKWARD, FFTW_EXHAUSTIVE);
#ifdef PAS_SCAL
         pfordealps = fftw_plan_dft_1d(xresdealps, indealps, outdealps, FFTW_FORWARD,
              FFTW_EXHAUSTIVE);
         pbackdealps = fftw_plan_dft_1d(xresdealps, indealps, outdealps,
             FFTW_BACKWARD, FFTW_EXHAUSTIVE);
         pchebfordealps = fftw_plan_dft_1d(2*(yresdealps-1), inchebdealps,
             outchebdealps, FFTW_FORWARD, FFTW_EXHAUSTIVE);
         pchebbackdealps = fftw_plan_dft_1d(2*(yresdealps-1), inchebdealps,
             outchebdealps, FFTW_BACKWARD, FFTW_EXHAUSTIVE);
#endif

if(rank == 0) printf("\nBoundary Vals initiated\n");
// Initialise boundaryvals=======================================================
         for(i = 0;i<xres;i++)
              {boundaryval1[i] = 1.0; boundaryval2[i] = 0.0;}
         for(i = xres;i<2*xres;i++)
              {boundaryval1[i] = 0.0; boundaryval2[i] = 1.0;}
         makezero(boundaryval0, 2*xres);

        for(i = 0; i<xres;i++)
                {if(i+xres/2 < xres)
                        {boundaryvalop[i] = boundaryval[i+xres/2];
                        boundaryvalop[i+xres] = boundaryval[i+xres/2+xres];
```

```
                        }
                else
                        {boundaryvalop[i] = boundaryval[i+xres/2-xres];
                        boundaryvalop[i+xres] = boundaryval[i+xres/2+xres -xres];
                        }
                }

//      if(rank == 0){printf("\nop= \n");print2Dmat(boundaryvalop,2,xres);}

// Assemble Influence Matrix for 1st order Backward Euler=======================
if(rank == 0) printf("\nAssemble Influence Matrix for 1st order Backward Euler\n");
         calcBEomegak12(omegak1, boundaryval1, Re, deltaTBE,in,out,pback,pfor,incheb,
             outcheb,pchebback,Lcheb,Lfour, yres-1,xres-1,n1, n2, rank, tasks,
             tasksnum, numtasks, numprocs, MPI_complex);
         calcBEomegak12(omegak2, boundaryval2, Re, deltaTBE,in,out,pback,pfor,incheb,
             outcheb,pchebback,Lcheb,Lfour, yres-1,xres-1,n1, n2, rank, tasks,
             tasksnum, numtasks, numprocs, MPI_complex);
         calcpsik(psik1, omegak1,Lcheb, Lfour, yres-1, xres-1,n1, n2, rank,tasks,
             tasksnum, numtasks,numprocs, MPI_complex);
         calcpsik(psik2, omegak2,Lcheb, Lfour, yres-1, xres-1,n1, n2, rank,tasks,
             tasksnum, numtasks,numprocs, MPI_complex);
if(rank == 0) printf("\n12 done\n");
        #ifdef DEBUG
        if(rank == 0)
        {chebbackcol2Dc(psik1,temp2D1,incheb,outcheb,pchebback,yres-1,xres-1,
            MPI_complex);
         printf("\nFour coeffs of psik1:\n");print2Dc(temp2D1,yres,xres);
         chebbackcol2Dc(psik2,temp2D1,incheb,outcheb,pchebback,yres-1,xres-1,
             MPI_complex);
         printf("\nFour coeffs of psik2:\n");print2Dc(temp2D1,yres,xres);
        }
        #endif

iterations = 0;
// Set boundaryvalt
        if(boundtime > 0.0)
                for(i = 0;i<xres;i++)
                        {boundaryvalt[i] = (boundaryval[i+xres] - boundaryval[i])
                            /2.0*sin(iterations*deltaTBE*2.0*PI/boundtime);
                        boundaryvalt[i+xres] = (-1.0) * boundaryvalt[i];
                        }

         asmbinfmat(Ainv, psik1, psik2,  in, out, pback, incheb, outcheb, pchebback,
             boundaryvalt, Lcheb,yres-1,xres-1,n1, n2, rank,tasks, tasksnum, numtasks
             ,numprocs, MPI_complex);

// Initialise omegak and psik===========================================
         initomegak(omegaknmin1, in, out, pfor, pback, incheb, outcheb, pchebfor,
             pchebback, boundaryvalt, Lcheb,yres-1, xres-1,n1, n2, rank,tasks,
             tasksnum, numtasks,numprocs, MPI_complex);
         calcpsik(psiknmin1, omegakn,Lcheb,Lfour, yres-1, xres-1,n1, n2, rank,tasks,
             tasksnum, numtasks,numprocs, MPI_complex);


#ifdef POINCARE
 if(rank == 0) readcoord(inputfpcare, &coordpcare, &coordnumpcare);
 MPI_Bcast(&coordnumpcare,1,MPI_UNSIGNED_LONG,0,MPI_COMM_WORLD);
 if(rank != 0) coordpcare = malloc(sizeof(double)*coordnumpcare*2);
 MPI_Bcast(coordpcare, coordnumpcare*2, MPI_DOUBLE, 0, MPI_COMM_WORLD);
 coordnpcare = malloc(sizeof(double)*coordnumpcare*2);
 copyarray(coordpcare, coordnpcare, coordnumpcare*2);

 coordfinalpcare = malloc(sizeof(double) * ((int)(time*(ylen/U))+1) * coordnumpcare
     *2);
```

```
 makezero(coordfinalpcare, ((int)(time*(ylen/U))+1) * coordnumpcare*2);
 copyarray(coordpcare , coordfinalpcare, coordnumpcare*2);
 coordfinalcountpcare =  coordfinalcountpcare +1.0;
 getchebu1kcol2Dc (psiknmin1, ukn, Lcheb, yres-1, xres-1);
 getfouru1krow2Dc (psiknmin1, vkn, Lfour, yres-1, xres-1);
 for(i = 0; i<(yres)*(xres); i++)
        {vkn[i][0] = (-1.0)*vkn[i][0]; vkn[i][1] = (-1.0)*vkn[i][1];}

 copycomplex(ukn, uk, yres*xres);
 copycomplex(vkn, vk, yres*xres);
#endif

#ifdef TRACKER
 if(rank == 0) readcoord(inputfpcare, &coordpcare, &coordnumpcare);
 MPI_Bcast(&coordnumpcare,1,MPI_UNSIGNED_LONG,0,MPI_COMM_WORLD);
 if(rank != 0) coordpcare = malloc(sizeof(double)*coordnumpcare*2);
 MPI_Bcast(coordpcare, coordnumpcare*2, MPI_DOUBLE, 0, MPI_COMM_WORLD);
 coordnpcare = malloc(sizeof(double)*coordnumpcare*2);
 copyarray(coordpcare, coordnpcare, coordnumpcare*2);

// coordfinalpcare = malloc(sizeof(double) * ((int)(time*(ylen/U))+1) * coordnumpcare
    *2);

// makezero(coordfinalpcare, ((int)(time*(ylen/U))+1) * coordnumpcare*2);
// copyarray(coordpcare , coordfinalpcare, coordnumpcare*2);

 getchebu1kcol2Dc (psiknmin1, ukn, Lcheb, yres-1, xres-1);
 getfouru1krow2Dc (psiknmin1, vkn, Lfour, yres-1, xres-1);
 for(i = 0; i<(yres)*(xres); i++)
        {vkn[i][0] = (-1.0)*vkn[i][0]; vkn[i][1] = (-1.0)*vkn[i][1];}

 copycomplex(ukn, uk, yres*xres);
 copycomplex(vkn, vk, yres*xres);
#endif




#ifdef PAS_SCAL
         calcuvRK4(psiknmin1, uRK4, uRK4i, vRK4, vRK4i, Lcheb, Lfour,rk4ratio *2*3+1,
              yres-1, xres-1);
         hscount=1;

        if(rank == 0)if(readinputps(psinitfile, xresps, yresps,theta1knmin1,
            theta2knmin1)) printf("\nThetas Initialised\n");else printf("\nInput file
             for PS NOT FOUND!!!\n");

        MPI_Bcast (theta1knmin1,xresps*yresps,MPI_complex,0,MPI_COMM_WORLD);
        MPI_Bcast (theta2knmin1,xresps*yresps,MPI_complex,0,MPI_COMM_WORLD);
#endif
         for(i = 0;i<coordy;i++)
                for(j = 0;j<coordx;j++)
                        {coord[2*(i*(coordx)+j)] = (j*1.0)/((coordx-1)*1.0)*(xlen/
                            ylen);//*xres/(xres*1.0-1.0);
                        coord[2*(i*(coordx)+j)+1] = (i*1.0)/((coordy-1)*1.0)*2.0
                            -1.0;
                        }


        #ifdef DEBUG
        if(rank == 0)
        {printf("\nAinv: \n"); print2Dc(Ainv,xres,4);
        printf("\nInitialised omegaknmin1:\n"); print2Dc(omegaknmin1, yres, xres);
```

```
        printf("\nInitialised psiknmin1:\n"); print2Dc(psiknmin1, yres, xres);
        }
        #endif

         copycomplex(omegaknmin1, omegakn,xres*yres);
         copycomplex(psiknmin1, psikn,xres*yres);

#ifdef PAS_SCAL
         copycomplex(theta1knmin1, theta1kn,xresps*yresps);
         copycomplex(theta2knmin1, theta2kn,xresps*yresps);

         copycomplex(theta1knmin1, theta1knmin2,xresps*yresps);
         copycomplex(theta2knmin1, theta2knmin2,xresps*yresps);
#endif
if (rank == 0) printf("\nInit Done!!\n");

if(numprocs>4)
initF2BE(psikn, omegakn, omegan, omeganmin1, NLn, NLnmin1, in, out, pfor, pback,
    incheb, outcheb, pchebfor, pchebback, indeal, outdeal, pfordeal, pbackdeal,
    inchebdeal, outchebdeal, pchebfordeal, pchebbackdeal, deltaT2BE, Re, Lcheb, Lfour
    , yresdeal-1, xresdeal-1, yres-1, xres-1, MPI_complex);

// Divide processors between Theta1 and Theta2
#ifdef PAS_SCAL
if(theta2flag ==1)
        {


 procarray = malloc(sizeof(int)*numprocs);
 for(i = 0;i<numprocs;i++)
        procarray[i] = i;

 MPI_Comm_group(MPI_COMM_WORLD, &worldgroup); // Extract the original group


// allocate procs for uv:===================
 retseglengths(procalloc_theta, procallocnum_theta, numprocs, 2);

        MPI_Group_incl(worldgroup, procallocnum_theta[0], &(procarray[procalloc_theta
            [0]]), &group_theta1);
        MPI_Group_incl(worldgroup, procallocnum_theta[1], &(procarray[procalloc_theta
            [1]]), &group_theta2);

        MPI_Comm_create(MPI_COMM_WORLD, group_theta1, &comm_theta1);
        MPI_Comm_create(MPI_COMM_WORLD, group_theta2, &comm_theta2);

 if(rank<procalloc_theta[0]+procallocnum_theta[0])
        { MPI_Group_rank (group_theta1, &rank_theta1);}
else
        { MPI_Group_rank (group_theta2, &rank_theta2);}

        getn1n2(&n1ps1, &n2ps1, rank_theta1,tasksps1, tasksnumps1, &numtasksps1,
            procallocnum_theta[0], ceil1(xresps,2)+1);
        getn1n2(&n1ps2, &n2ps2, rank_theta2,tasksps2, tasksnumps2, &numtasksps2,
            procallocnum_theta[1], ceil1(xresps,2)+1);

        }
else
        {rank_theta1 = rank;
        comm_theta1 = MPI_COMM_WORLD;
        procallocnum_theta[0] = numprocs;
        getn1n2(&n1ps1, &n2ps1, rank_theta1,tasksps1, tasksnumps1, &numtasksps1,
            procallocnum_theta[0], ceil1(xresps,2)+1);
        }
```

```
#endif
// INIT DONE!!========================================================================


if(rank == 0) printf("\nParameters output started\n");
// Print problem parameters ===========================================================
if(rank ==0)
        {       strcpy(outstr1,outstr);
//              strcat(outstr1,"N");
                strcat(outstr1,".m");
                printf("\nDESCRIPTION = %s\n",description);
                printf("\noutstr = %s\n",outstr1);
                outputTfp = fopen(outstr1,"w");
                fprintf(outputTfp,"\n%% Re = %g\n",Re);
//              fprintf(outputTfp,"\nTIME = %g [s]\n",iterations*deltaT2BE*(ylen/U))
    ;
                fprintf(outputTfp,"\n%%DESCRIPTION = %s\n",description);
                fprintf(outputTfp,"\n%%TOTALTIME = %g [s]\n",time*(ylen/U));
                fprintf(outputTfp,"\n%%DELTAT = %g\n",deltaT2BE);
                fprintf(outputTfp,"\n%%DELTATUV = %g\n",deltaTuv);
                fprintf(outputTfp,"\n%%DELTATPS = %g\n",deltaTps);
                fprintf(outputTfp,"\n%%HS = %g\n",hs);
                fprintf(outputTfp,"\n%%OUTPERIOD = %g\n",outtime);
                fprintf(outputTfp,"\n%%BOUNDPERIOD = %g [s]\n",boundtime*(ylen/U));
                fprintf(outputTfp,"\n%%SWITCHPERIOD = %g [s]\n",switchtime*(ylen/U))
                    ;
                fprintf(outputTfp,"\n%%REVERSEPERIOD = %g [s]\n",reversetime*(ylen/U
                    ));
                fprintf(outputTfp,"\n%% SIZE : %d %d\n",xres,yres);
                fprintf(outputTfp,"\n%% PSRES : %d %d\n",xresps,yresps);
                fprintf(outputTfp,"\nSC1 = %16.16g\n",Sc1);
                fprintf(outputTfp,"\nSC2 = %16.16g\n",Sc2);
                fprintf(outputTfp,"\n%% LENGTHS : %16.16g %16.16g\n",xlen,ylen);
                fprintf(outputTfp,"\nUMAX = %16.16g\n",U);
                fprintf(outputTfp,"\n\n%%FOR MATLAB:\n");
                fprintf(outputTfp,"\ncoordx = %d\n",coordx);
                fprintf(outputTfp,"\ncoordy = %d\n",coordy);
                fprintf(outputTfp,"\nR = %g;\n",Re);
                fprintf(outputTfp,"\ndeltaT2BE = %g\n",deltaT2BE);
                fprintf(outputTfp,"\noutputsperfile = %g\n",outputsperfile);
                fprintf(outputTfp,"\nzerovelflag = %d\n",zerovelflag);
#ifdef POINCARE
                fprintf(outputTfp,"\ncoordnumpcare = %lu\n",coordnumpcare);
                fprintf(outputTfp,"\ncoordnum = %lu\n",((int)(time*(ylen/U))+1) *
                    coordnumpcare);
#endif
#ifdef TRACKER
                fprintf(outputTfp,"\ncoordnum = %lu\n",coordnumpcare);
#endif
                fprintf(outputTfp,"\nxres = %d;\nyres = %d;\n",xres,yres);
#ifdef PAS_SCAL
                fprintf(outputTfp,"\nxresps = %d;\nyresps = %d;\n",xresps,yresps);
#endif
                fprintf(outputTfp,"\nxlen = %16.16g;\nylen = %16.16g;\n\n\n",xlen,
                    ylen);
                fprintf(outputTfp,"\n%%file output timestep in seconds:\ntimestep =
                    %16.16g;\n\n",outtime*(ylen/U));
               fclose(outputTfp);
        }
        //fprintf(output2fp,"\nu and v from initialised psikn\n");fprintuv(output2fp,
            psikn, in, out, pback,incheb, outcheb, pchebback,Lcheb,Lfour,yres -1,
            xres -1);
```

```
                                                                                805
                                                                                806
//      if(iterations*deltaTBE == outputcount*outtime || iterations*deltaTBE >    807
    outputcount*outtime )
iterations = 0;                                                                 808
                {printpsiuv(psikn, &outputcount, outputsperfile, outstr, outputoption  809
                    , valoutput, coordx, coordy, coordn, coord, Lcheb, Lfour, yres-1,
                     xres-1, rank, numprocs,&eventflag, iterations*deltaTBE,
                    iterations*deltaTBE*ylen/U , MPI_complex);}
#ifdef PAS_SCAL                                                                 810
//      if(iterations*deltaTBE == outputcountps*outtimeps || iterations*deltaTBE >  811
    outputcountps*outtimeps )
//              MPI_Sendrecv (theta2kn,yresps*xresps,MPI_complex,0,1,theta2kn,yresps*  812
    xresps,MPI_complex,procalloc_theta[1],1,MPI_COMM_WORLD,&Stat);
                                                                                813
                {//printtheta(theta1kn, theta2kn, &outputcountps, outputsperfile,  814
                    outstr, outputoption, valoutput, coordx, coordy, coordn, coord,
                    Lcheb, Lfour, yresps-1, xresps-1, rank, numprocs,&eventflag,
                    iterations*deltaTBE, iterations*deltaTBE*ylen/U , MPI_complex);
                printtheta2(theta1kn, theta2kn, &outputcountps, outputsperfile,  815
                    outstr, outputoption, Lcheb, Lfour,yresps-1, xresps-1, rank,
                    numprocs, &eventflag, iterations*deltaTBE, iterations*deltaTBE*
                    ylen/U, pbackps, inps, outps, pchebbackps, inchebps, outchebps,
                      MPI_complex);
                }                                                               816
#endif                                                                          817
                                                                                818
#ifdef TRACKER                                                                  819
//      if(iterations*deltaT2BE == coordouttimepcare * coordoutcountpcare ||    820
    iterations*deltaT2BE > coordouttimepcare * coordoutcountpcare)
                {printcoord(coordpcare, &coordoutcountpcare, outputsperfile, outstr,  821
                    outputoption, coordnumpcare, rank, iterations*deltaT2BE,
                    iterations*deltaT2BE*ylen/U);
                        if(rank == 0)                                           822
                          {end = clock();                                       823
                          dif = ((double) (end - start)) / CLOCKS_PER_SEC;      824
                          printf("\n Tracker file no: %g,  TIME TAKEN = %gs, time = %  825
                              g, dim time = %gs\n",coordoutcountpcare-1, dif,
                              iterations*deltaT2BE, iterations*deltaT2BE*ylen/U);
                         }                                                      826
                }                                                               827
#endif                                                                          828
                                                                                829
                                                                                830
// Iterations for 1st Order Backward Euler=====================================  831
// ============================================================================  832
if(rank ==0) printf("\n Iterations for 1st Order BE started:\n");               833
//      if(time == 0.0)                                                         834
                iter = (int)(3.0/3.0*deltaT2BE/deltaTBE);                       835
//      else if(time<3.0/3.0*deltaT2BE)                                         836
//                      iter = (int)(time/deltaTBE);                            837
//              else iter = (int)(3.0/3.0*deltaT2BE/deltaTBE);                  838
                                                                                839
        for (iterations = 1,itconv = 0;iterations <iter+1; iterations++)        840
                {if(rank == 0)if(iterations/10==iterations/10.0)                841
                  {end = clock();                                               842
                  dif = ((double) (end - start)) / CLOCKS_PER_SEC;              843
                  printf("\niteration: %lu\t\tprocess# %d: TIME TAKEN = %gs \t\t Non  844
                      dim time = %g \t\t dim time = %gs\n",iterations,rank,dif,
                      iterations*deltaTBE, iterations*deltaTBE*(ylen/U));
                #ifdef PAS_SCAL                                                 845
                  printf("\nFor theta1: ");maxc(theta1k,xresps*yresps);           846
                #endif                                                          847
                  printf("\n#%d: For psi: ",rank);maxc(psik,xres*yres);         848
```

```
                  }//printf("|%d|",rank);printf("\n!!!\n");                                    849
                if(rank==0)if(iterations==0||iterations==1||iterations==2||iterations        850
                    ==3)printf("!!ITERATION: %lu\n",iterations);
                                                                                             851
// Set boundaryvalt                                                                         852
        if(boundtime > 0.0)                                                                  853
                {for(i = 0;i<xres;i++)                                                       854
                        {boundaryvalt[i] = (boundaryval[i+xres] - boundaryval[i])            855
                            /2.0*sin(iterations*deltaTBE*2.0*PI/boundtime);
                        boundaryvalt[i+xres] = (-1.0) * boundaryvalt[i];                     856
                        }                                                                    857
                 asmbinfmat(Ainv, psik1, psik2,  in, out, pback, incheb, outcheb,            858
                     pchebback,boundaryvalt, Lcheb,yres-1,xres-1,n1, n2, rank,tasks,
                     tasksnum, numtasks,numprocs, MPI_complex);
                }                                                                            859
                                                                                             860
        if(reversetime > 0.0)                                                                861
        if(((iterations-1)*deltaTBE == reversetime ||(iterations-1)*deltaTBE >               862
            reversetime) && reversecount == 0)
                {for(i = 0;i<xres*2;i++)                                                     863
                                {boundaryvalt[i] = -boundaryvalt[i];                         864
                                boundaryvalop[i] = -boundaryvalop[i];                        865
                                boundaryval[i] = -boundaryval[i];                            866
                                }                                                            867
                                                                                             868
                 asmbinfmat(Ainv, psik1, psik2,  in, out, pback, incheb, outcheb,            869
                     pchebback,boundaryvalt, Lcheb,yres-1,xres-1,n1, n2, rank,tasks,
                     tasksnum, numtasks,numprocs, MPI_complex);
                switchcount++;itconv =0;  eventflag = 1;                                     870
                if(rank == 0) printf("\nBoundary Velocities reversed at t = %16.16g [       871
                    s]\n",iterations*deltaTBE*(ylen/U));
                reversecount++;                                                              872
                }                                                                            873
                                                                                             874
        if(switchtime > 0.0)                                                                 875
        if((iterations-1)*deltaTBE == switchcount*switchtime ||(iterations-1)*               876
            deltaTBE > switchcount*switchtime)
                {if(phase == 0)                                                              877
                        {phase = 1;                                                          878
                        for(i = 0;i<xres*2;i++)                                              879
                                {boundaryvalt[i] = boundaryvalop[i];                         880
                                }                                                            881
                        }                                                                    882
                else                                                                         883
                        {phase = 0;                                                          884
                        for(i = 0;i<xres*2;i++)                                              885
                                {boundaryvalt[i] = boundaryval[i];                           886
                                }                                                            887
                        }                                                                    888
                 asmbinfmat(Ainv, psik1, psik2,  in, out, pback, incheb, outcheb,            889
                     pchebback,boundaryvalt, Lcheb,yres-1,xres-1,n1, n2, rank,tasks,
                     tasksnum, numtasks,numprocs, MPI_complex);
                switchcount++;itconv =0; eventflag = 1;                                      890
                if(rank == 0) printf("\nBoundary Velocities switched at t = %16.16g [       891
                    s]\n",iterations*deltaTBE*(ylen/U));
                }                                                                            892
                                                                                             893
                                                                                             894
                                                                                             895
// Calculate uv ================================================================            896
              calcBEomegaktilde(omegaktilde,  psikn, omegakn, in, out, pfor, pback,          897
                  incheb, outcheb, pchebfor, pchebback, indeal, outdeal, pfordeal,
                  pbackdeal, inchebdeal, outchebdeal, pchebfordeal, pchebbackdeal ,
                  deltaTBE, Re, Lcheb, Lfour,yresdeal-1, xresdeal-1, yres-1, xres-1,
```

```
                n1, n2, rank,tasks, tasksnum, numtasks,numprocs, MPI_complex);
               calcpsik(psiktilde, omegaktilde,Lcheb, Lfour, yres-1,xres-1,n1, n2,    898
                   rank,tasks, tasksnum, numtasks,numprocs, MPI_complex);
               calclambda(lambda, psiktilde, Ainv,in, out, pfor,pback, incheb,        899
                   outcheb, pchebback, boundaryvalt, Lcheb,yres-1, xres-1,n1, n2,
                   rank,tasks, tasksnum, numtasks,numprocs, MPI_complex);
            calcomegak(omegak, omegaktilde, omegak1, omegak2, lambda, incheb,         900
                outcheb,pchebfor,pchebback,yres -1, xres -1,n1, n2, rank,tasks,
                tasksnum, numtasks,numprocs, MPI_complex);
            calcomegak(psik, psiktilde,psik1, psik2, lambda, incheb,outcheb,          901
                pchebfor,pchebback,yres-1, xres-1,n1, n2, rank,tasks, tasksnum,
                numtasks,numprocs, MPI_complex);
//=============================================================================       902
                                                                                      903
#ifdef PAS_SCAL                                                                       904
               if(iterations*deltaTBE == hscount*hs/2 || iterations*deltaTBE>hscount   905
                   *hs/2)
               {calcuvRK4(psik, uRK4, uRK4i, vRK4, vRK4i, Lcheb, Lfour, rk4ratio      906
                   *2*3+1, yres-1, xres-1);
                hscount++;                                                            907
//             if(rank == 0) printf("\nhscount = %d, time = %g\n",hscount,iterations   908
     * deltaTBE);
               }                                                                      909
                                                                                      910
                changeres(psikn,psipskn,yres,xres,yresps,xresps);                     911
               if(rank_theta1 != -1)                                                  912
                calcpsBE(theta1k, psipskn, theta1kn, inps, outps, pforps, pbackps,    913
                    inchebps, outchebps, pchebforps, pchebbackps, indealps,
                    outdealps, pfordealps, pbackdealps, inchebdealps, outchebdealps,
                     pchebfordealps, pchebbackdealps, deltaTBE, Re, Sc1, Lcheb,
                    Lfour, yresdealps-1, xresdealps-1, yresps-1, xresps-1, n1ps1,
                    n2ps1, rank_theta1, tasksps1, tasksnumps1, numtasksps1,
                    procallocnum_theta[0],comm_theta1, MPI_complex);
               if(rank_theta2 != -1)                                                  914
               if(theta2flag ==1)                                                     915
               calcpsBE(theta2k, psipskn, theta2kn, inps, outps, pforps, pbackps,     916
                   inchebps, outchebps, pchebforps, pchebbackps, indealps, outdealps
                   , pfordealps, pbackdealps, inchebdealps, outchebdealps,
                   pchebfordealps, pchebbackdealps, deltaTBE, Re, Sc2, Lcheb, Lfour,
                    yresdealps-1, xresdealps-1, yresps-1, xresps-1, n1ps2, n2ps2,
                   rank_theta2, tasksps2, tasksnumps2, numtasksps2,
                   procallocnum_theta[1], comm_theta2, MPI_complex);
#endif                                                                                917
                                                                                      918
        #ifdef DEBUG                                                                  919
        if(rank == 0)                                                                 920
             {chebbackcol2Dc(omegaktilde,temp2D1,incheb,outcheb,pchebback,yres-1,xres 921
                 -1, MPI_complex);
              printf("\nFourier Coeffs of Omegaktilde: \n");print2Dc(temp2D1,yres,    922
                  xres);
             chebbackcol2Dc(psiktilde,temp2D1,incheb,outcheb,pchebback,yres-1,xres-1, 923
                  MPI_complex);
              printf("\nFourier Coeffs of psiktilde: \n");print2Dc(temp2D1,yres,     924
                  xres);
             printf("\nLambda:%d\n",iterations);print2Dc(lambda,2,xres);             925
              printf("\nomegaktilde\n"); print2Dc(omegaktilde,yres,xres);            926
              printf("\nomegak\n"); print2Dc(omegak,yres,xres);                      927
              printf("\npsiktilde\n"); print2Dc(psiktilde,yres,xres);                928
              printf("\npsik\n"); print2Dc(psik,yres,xres);                          929
              printf("\n\nIteration: %d\n\n",iterations);                           930
              }                                                                       931
        #endif                                                                        932
                                                                                      933
         if(itconv == 0)                                                              934
```

```
                if(checkconvc(psik,psikn,ERROR,xres*yres))
                        {itconv = iterations;
                        if(time ==0.0)
                                flag = 1.0;
                        }

         if(flag) break;

              copycomplex(omegak, omegakn,xres*yres);
              copycomplex(psik, psikn,xres*yres);
#ifdef PAS_SCAL
              copycomplex(theta1k, theta1kn,xresps*yresps);
              copycomplex(theta2k, theta2kn,xresps*yresps);
#endif
                #ifdef DEBUG
                if(rank == 0){   printf("\n!!!!!!!!!!!!!!!!!!!!!!!!\n");
                         printuv(psik, in, out, pback,incheb, outcheb, pchebback,
                             Lcheb,Lfour,yres -1, xres -1)
                             ;//////////////////////////////////////
                                 }
                #endif

//Prints out output periodically=====================================================
        if(iterations*deltaTBE == outputcount*outtime || iterations*deltaTBE >
            outputcount*outtime )
                {printpsiuv(psik, &outputcount,outputsperfile, outstr, outputoption,
                    valoutput, coordx, coordy, coordn, coord, Lcheb, Lfour, yres-1,
                    xres-1, rank, numprocs,&eventflag, iterations*deltaTBE,
                    iterations*deltaTBE*ylen/U , MPI_complex);}
#ifdef PAS_SCAL
        if(iterations*deltaTBE == outputcountps*outtimeps || iterations*deltaTBE >
            outputcountps*outtimeps )
                {MPI_Sendrecv (theta2kn,yresps*xresps,MPI_complex,0,1,theta2kn,yresps
                    *xresps,MPI_complex,procalloc_theta[1],1,MPI_COMM_WORLD,&Stat);

                //printtheta(theta1k, theta2k, &outputcountps, outputsperfile, outstr
                    , outputoption, valoutput, coordx, coordy, coordn, coord, Lcheb,
                    Lfour, yresps-1, xresps-1, rank, numprocs,&eventflag, iterations*
                    deltaTBE, iterations*deltaTBE*ylen/U , MPI_complex);
                printtheta2(theta1k, theta2k, &outputcountps, outputsperfile, outstr,
                     outputoption, Lcheb, Lfour,yresps-1, xresps-1, rank,  numprocs,
                    &eventflag, iterations*deltaTBE, iterations*deltaTBE*ylen/U,
                    pbackps, inps, outps, pchebbackps, inchebps, outchebps,
                    MPI_complex);
                }
#endif


// Print output done ============================================================

        }

#ifdef POINCARE
 getchebu1kcol2Dc (psikn, ukint, Lcheb, yres-1, xres-1);
 getfouru1krow2Dc (psikn, vkint, Lfour, yres-1, xres-1);
 for(i = 0; i<(yres)*(xres); i++)
        {vkint[i][0] = (-1.0)*vkint[i][0]; vkint[i][1] = (-1.0)*vkint[i][1];}
#endif

#ifdef TRACKER
 getchebu1kcol2Dc (psikn, ukint, Lcheb, yres-1, xres-1);
 getfouru1krow2Dc (psikn, vkint, Lfour, yres-1, xres-1);
 for(i = 0; i<(yres)*(xres); i++)
        {vkint[i][0] = (-1.0)*vkint[i][0]; vkint[i][1] = (-1.0)*vkint[i][1];}
```

```
#endif

//Iterations for 1st order Backward Euler done! ================================
// ============================================================================

#ifdef PAS_SCAL
          if(rank == 0){maxc(psik,yres*xres); printf("\n#%d: For theta1: ",rank);maxc
              (theta1k,xresps*yresps);}
#endif

        #ifdef DEBUG
         if(rank == 0)
        {printuv(psik, in, out, pback,incheb, outcheb, pchebback, Lcheb,Lfour,yres
            -1, xres -1, MPI_complex);

         chebbackcol2Dc(psik,temp2D1,incheb,outcheb,pchebback,yres -1, xres -1,
             MPI_complex);
         fourbackrow2Dc(temp2D1,temp2D2,in,out,pback,yres -1, xres -1, MPI_complex);
         printf("\npsi:\n");print2Dc(temp2D2,yres,xres);

         chebbackcol2Dc(omegak,temp2D1,incheb,outcheb,pchebback,yres -1, xres -1,
             MPI_complex);
         fourbackrow2Dc(temp2D1,temp2D2,in,out,pback,yres -1, xres -1, MPI_complex);
         printf("\nomega:\n");print2Dc(temp2D2,yres,xres);

         printf("\npsik:\n");print2Dc(psik,yres,xres);
         printf("\nomegak:\n");print2Dc(omegak,yres,xres);

         getfouru1krow2Dc (omegak, temp2D2,Lfour,yres-1,  xres-1);
         printf("\nomegak(1):\n");print2Dc(temp2D2,yres,xres);
         chebbackcol2Dc(temp2D2,temp2D1,incheb,outcheb,pchebback,yres -1, xres -1,
             MPI_complex);
         fourbackrow2Dc(temp2D1,temp2D2,in,out,pback,yres -1, xres -1, MPI_complex);
         printf("\nomega(1):\n");print2Dc(temp2D2,yres,xres);
        }
        #endif
        if(rank == 0)
                {if (itconv ==0)
                        printf("\nConvergence not achieved.\n");
                else
                        if(rank == 0) printf("\n Convergence achieved at iteration =
                            %lu & time = %g\n",itconv, deltaTBE*itconv*(ylen/U));
                }

// Assemble Influence Matrix for 2nd order Backward Euler=======================
             calc2BEomegak12(omegak1, boundaryval1, Re, deltaT2BE,in,out,pback,pfor,
                 incheb,outcheb,pchebback,Lcheb,Lfour, yres-1,xres-1,n1, n2, rank,
                 tasks, tasksnum, numtasks,numprocs, MPI_complex);
             calc2BEomegak12(omegak2, boundaryval2, Re, deltaT2BE,in,out,pback,pfor,
                 incheb,outcheb,pchebback,Lcheb,Lfour, yres-1,xres-1,n1, n2, rank,
                 tasks, tasksnum, numtasks,numprocs, MPI_complex);
             calcpsik(psik1, omegak1,Lcheb, Lfour, yres-1, xres-1,n1, n2, rank,tasks,
                 tasksnum, numtasks,numprocs, MPI_complex);
             calcpsik(psik2, omegak2,Lcheb, Lfour, yres-1, xres-1,n1, n2, rank,tasks,
                 tasksnum, numtasks,numprocs, MPI_complex);

// Set boundaryvalt
        if(boundtime > 0.0)
                for(i = 0;i<xres;i++)
                        {boundaryvalt[i] = (boundaryval[i+xres] - boundaryval[i])
                            /2.0*sin(iterations*deltaTBE*2.0*PI/boundtime);
                        boundaryvalt[i+xres] = (-1.0) * boundaryvalt[i];
                        }
```

```
            asmbinfmat(Ainv, psik1, psik2,  in, out, pback, incheb, outcheb,
                pchebback,boundaryvalt, Lcheb,yres-1,xres-1,n1, n2, rank,tasks,
                tasksnum, numtasks,numprocs, MPI_complex);

        #ifdef DEBUG
        if(rank == 0)
                {chebbackcol2Dc(psik1,temp2D1,incheb,outcheb,pchebback,yres-1,xres-1,
                     MPI_complex);
                printf("\nFour coeffs of psik1:\n");print2Dc(temp2D1,yres,xres);
                chebbackcol2Dc(psik2,temp2D1,incheb,outcheb,pchebback,yres-1,xres-1,
                    MPI_complex);
                printf("\nFour coeffs of psik2:\n");print2Dc(temp2D1,yres,xres);
             printf("\nAinv: \n"); print2Dc(Ainv,xres,4);
                }
        #endif

// end asmb influence matrix ================================================


// Iterations for 2nd Order Backward Euler==================================
// ========================================================================

        iter = (int)(time/deltaT2BE);
        if(rank == 0)printf("\n 2nd order BE: iter = %lu\n",iter);

#ifdef PAS_SCAL
        iter = (int)(2.0/1.0*deltaTps/deltaT2BE);
        if(rank == 0)printf("\n 2nd order BE: iter = %lu\n",iter);
#endif
        if (flag);
        else
        {for (iterations = 2,itconv = 0; iterations <iter + 1; iterations++)
                {if(rank==0)if(iterations/10000==iterations/10000.0)
                  {end = clock();
                  dif = ((double) (end - start)) / CLOCKS_PER_SEC;
                  printf("\niteration: %lu\t\tprocess# %d: TIME TAKEN = %gs \t\t Non
                      dim time = %g \t\t dim time = %gs\n",iterations,rank,dif,
                      iterations*deltaT2BE, iterations*deltaT2BE*(ylen/U));
                #ifdef PAS_SCAL
                  printf("\nFor theta1: ");maxc(theta1k,xresps*yresps);
                #endif
                                printf("\n#%d: For psi: ",rank);maxc(psik,xres*yres
                                    );
                }
// Set boundaryvalt
        if(boundtime > 0.0)
                {for(i = 0;i<xres;i++)
                        {boundaryvalt[i] = (boundaryval[i+xres] - boundaryval[i])
                            /2.0*sin(iterations*deltaTBE*2.0*PI/boundtime);
                        boundaryvalt[i+xres] = (-1.0) * boundaryvalt[i];
                        }
                 asmbinfmat(Ainv, psik1, psik2,  in, out, pback, incheb, outcheb,
                     pchebback,boundaryvalt, Lcheb,yres-1,xres-1,n1, n2, rank,tasks,
                     tasksnum, numtasks,numprocs, MPI_complex);
                }

        if(reversetime > 0.0)
        if(((iterations-1)*deltaT2BE == reversetime || (iterations-1)*deltaT2BE >
            reversetime) && reversecount == 0)
                {for(i = 0;i<xres*2;i++)
                                {boundaryvalt[i] = -boundaryvalt[i];
                                boundaryvalop[i] = -boundaryvalop[i];
                                boundaryval[i] = -boundaryval[i];
                                }
```

```
                                                                                1088
                 asmbinfmat(Ainv, psik1, psik2,  in, out, pback, incheb, outcheb,      1089
                     pchebback,boundaryvalt, Lcheb,yres-1,xres-1,n1, n2, rank,tasks,
                     tasksnum, numtasks,numprocs, MPI_complex);
                switchcount++;itconv =0;  eventflag = 1;                        1090
                if(rank == 0) printf("\nBoundary Velocities reversed at t = %16.16g [  1091
                    s]\n",iterations*deltaT2BE*(ylen/U));
                reversecount++;                                                 1092
                }                                                               1093
                                                                                1094
        if(switchtime > 0.0)                                                    1095
        if((iterations-1)*deltaT2BE == switchcount*switchtime || (iterations-1)*  1096
            deltaT2BE > switchcount*switchtime)
                {if(phase == 0)                                                 1097
                        {phase = 1;                                             1098
                        for(i = 0;i<xres*2;i++)                                 1099
                                {boundaryvalt[i] = boundaryvalop[i];            1100
                                }                                               1101
                        }                                                       1102
                else                                                            1103
                        {phase = 0;                                             1104
                        for(i = 0;i<xres*2;i++)                                 1105
                                {boundaryvalt[i] = boundaryval[i];              1106
                                }                                               1107
                        }                                                       1108
                 asmbinfmat(Ainv, psik1, psik2,  in, out, pback, incheb, outcheb,      1109
                     pchebback,boundaryvalt, Lcheb,yres-1,xres-1,n1, n2, rank,tasks,
                     tasksnum, numtasks,numprocs, MPI_complex);
                switchcount++;itconv =0;  eventflag = 1;                        1110
#ifndef POINCARE                                                                1111
                if(rank == 0) printf("\nBoundary Velocities switched at t = %16.16g [  1112
                    s]\n",iterations*deltaT2BE*(ylen/U));
#endif                                                                          1113
                }                                                               1114
                                                                                1115
                                                                                1116
// Calculate uv ================================================================  1117
                      //calc2BEomegaktilde(omegaktilde,  psikn,psiknmin1, omegakn,      1118
                          omegaknmin1, in, out, pfor, pback, incheb, outcheb,
                          pchebfor, pchebback, indeal, outdeal, pfordeal, pbackdeal,
                          inchebdeal, outchebdeal, pchebfordeal, pchebbackdeal,
                          deltaT2BE, Re, Lcheb, Lfour, yresdeal-1, xresdeal-1, yres
                          -1, xres-1,n1, n2, rank,tasks, tasksnum, numtasks,numprocs,
                           MPI_complex);
                         calc2BEomegaktilde(omegaktilde, psikn, omegakn, omegan,        1119
                             omeganmin1, NLn, NLnmin1, psiknmin1, omegaknmin1, in, out
                             , pfor, pback, incheb, outcheb, pchebfor, pchebback,
                             indeal, outdeal, pfordeal, pbackdeal, inchebdeal,
                             outchebdeal, pchebfordeal, pchebbackdeal, deltaT2BE, Re,
                             Lcheb, Lfour, yresdeal-1, xresdeal-1, yres-1, xres-1, n1,
                              n2, rank, tasks, tasksnum, numtasks, numprocs,
                             MPI_complex);
                          calcpsik(psiktilde, omegaktilde,Lcheb, Lfour, yres-1,xres-1,  1120
                              n1, n2, rank,tasks, tasksnum, numtasks,numprocs,
                              MPI_complex);
                         calclambda(lambda, psiktilde, Ainv,in, out, pfor,pback,        1121
                              incheb, outcheb, pchebback, boundaryvalt, Lcheb,yres-1,
                              xres-1,n1, n2, rank,tasks, tasksnum, numtasks,numprocs,
                              MPI_complex);
                 calcomegak(omegak, omegaktilde, omegak1, omegak2, lambda, incheb,      1122
                     outcheb,pchebfor,pchebback,yres -1, xres -1,n1, n2, rank,tasks,
                     tasksnum, numtasks,numprocs, MPI_complex);
                 calcomegak(psik, psiktilde,psik1, psik2, lambda, incheb,outcheb,       1123
                     pchebfor,pchebback,yres-1, xres-1,n1, n2, rank,tasks, tasksnum,
```

```
                    numtasks,numprocs, MPI_complex);
// ========================================================================
#ifdef PAS_SCAL
                if(iterations*deltaT2BE == hscount*hs/2 || iterations*deltaT2BE>
                    hscount*hs/2)
                {calcuvRK4(psik, uRK4, uRK4i, vRK4, vRK4i, Lcheb, Lfour, rk4ratio
                    *2*3+1, yres-1, xres-1);
                 hscount++;
                }

                changeres(psikn,psipskn,yres,xres,yresps,xresps);
                changeres(psiknmin1,psipsknmin1,yres,xres,yresps,xresps);
                        if(rank_theta1 != -1)
                        calcps2BE(theta1k, psipskn, psipsknmin1, theta1kn,
                            theta1knmin1, inps, outps, pforps, pbackps, inchebps,
                            outchebps, pchebforps, pchebbackps, indealps, outdealps,
                            pfordealps, pbackdealps, inchebdealps, outchebdealps,
                            pchebfordealps, pchebbackdealps, deltaT2BE, Re, Sc1,
                            Lcheb, Lfour,  yresdealps -1 , xresdealps - 1,yresps -1 ,
                             xresps - 1, n1ps1, n2ps1, rank_theta1, tasksps1,
                            tasksnumps1, numtasksps1, procallocnum_theta[0],
                            comm_theta1, MPI_complex);
                if(rank_theta2 != -1)
                if(theta2flag ==1)
                        calcps2BE(theta2k, psipskn, psipsknmin1, theta2kn,
                            theta2knmin1, inps, outps, pforps, pbackps, inchebps,
                            outchebps, pchebforps, pchebbackps, indealps, outdealps,
                            pfordealps, pbackdealps, inchebdealps, outchebdealps,
                            pchebfordealps, pchebbackdealps, deltaT2BE, Re, Sc2,
                            Lcheb, Lfour, yresdealps -1 , xresdealps - 1, yresps -1 ,
                             xresps - 1, n1ps2, n2ps2, rank_theta2, tasksps2,
                            tasksnumps2, numtasksps2, procallocnum_theta[1],
                            comm_theta2, MPI_complex);
#endif

        #ifdef DEBUG
        if(rank == 0)
                {       chebbackcol2Dc(omegaktilde,temp2D1,incheb,outcheb,pchebback,
                    yres-1,xres-1, MPI_complex);
                        printf("\nFourier Coeffs of Omegaktilde: \n");print2Dc(
                            temp2D1,yres,xres);
                chebbackcol2Dc(psiktilde,temp2D1,incheb,outcheb,pchebback,yres-1,xres
                    -1, MPI_complex);
                        printf("\nFourier Coeffs of psiktilde: \n");print2Dc(temp2D1,
                            yres,xres);
                printf("\nLambda:%d\n",iterations);print2Dc(lambda,2,xres);
                        printf("\nomegaktilde\n"); print2Dc(omegaktilde,yres,xres);
                        printf("\nomegak\n"); print2Dc(omegak,yres,xres);
                        printf("\npsiktilde\n"); print2Dc(psiktilde,yres,xres);
                        printf("\npsik\n"); print2Dc(psik,yres,xres);
                        printf("\n\nIteration: %d\n\n",iterations);
                }
        #endif

#ifndef POINCARE
                        if(itconv == 0)
                        if(checkconvc(psik,psikn,ERROR,xres*yres))
                                {itconv = iterations;
                                if(time ==0.0)
                                        flag = 1.0;
                                else
                                        if(rank == 0) printf("\n Convergence achieved
```

```
                                                at iteration = %lu & time = %16.16g\n",
                                               iterations, deltaT2BE*itconv*(ylen/U));
                                }                                                                    1165
                if(flag) break;                                                                      1166
#endif                                                                                               1167
                 copycomplex(omegakn, omegaknmin1,xres*yres);                                        1168
                 copycomplex(psikn, psiknmin1,xres*yres);                                            1169
                 copycomplex(omegak, omegakn,xres*yres);                                             1170
                 copycomplex(psik, psikn,xres*yres);                                                 1171
#ifdef PAS_SCAL                                                                                      1172
                 copycomplex(theta1kn, theta1knmin1,xresps*yresps);                                  1173
                 copycomplex(theta1k, theta1kn,xresps*yresps);                                       1174
                 copycomplex(theta2kn, theta2knmin1,xresps*yresps);                                  1175
                 copycomplex(theta2k, theta2kn,xresps*yresps);                                       1176
#endif                                                                                               1177
                                                                                                     1178
        #ifdef DEBUG                                                                                 1179
        if(rank == 0)                                                                                1180
                        {printf("\n!!!!!!!!!!!!!!!!!!!!!!!!\n");                                     1181
                        printuv(psik, in, out, pback,incheb, outcheb, pchebback,                     1182
                            Lcheb,Lfour,yres -1, xres -1, MPI_complex)
                            ;////////////////////////////////////
                        }                                                                            1183
        #endif                                                                                       1184
        if(rank == 0);                                                                               1185
                                                                                                     1186
//Prints out output periodically=================================================                   1187
        if(fabs(iterations*deltaT2BE - outputcount*outtime)<1e-10 || iterations*                     1188
            deltaT2BE > outputcount*outtime )
                printpsiuv(psik, &outputcount, outputsperfile, outstr, outputoption,                 1189
                    valoutput, coordx, coordy, coordn, coord, Lcheb, Lfour, yres-1,
                    xres-1, rank, numprocs, &eventflag, iterations*deltaT2BE,
                    iterations*deltaT2BE*ylen/U, MPI_complex);
#ifdef PAS_SCAL                                                                                      1190
        if(fabs(iterations*deltaT2BE - outputcountps*outtimeps)<1e-10 || iterations*                 1191
            deltaT2BE > outputcountps*outtimeps )
                {       if (rank == procalloc_theta[1])                                              1192
                                {MPI_Send(theta2k, yresps*xresps, MPI_complex, 0,                    1193
                                    150,MPI_COMM_WORLD);}
                        else if (rank == 0)                                                          1194
                                {MPI_Recv(theta2k, yresps*xresps, MPI_complex,                       1195
                                    procalloc_theta[1], 150, MPI_COMM_WORLD,&Stat);}
//MPI_Sendrecv (theta2kn,yresps*xresps,MPI_complex,0,1,theta2kn,yresps*xresps,                       1196
    MPI_complex,procalloc_theta[1],1,MPI_COMM_WORLD,&Stat);
                //printtheta(theta1k, theta2k, &outputcountps, outputsperfile, outstr                1197
                    , outputoption, valoutput, coordx, coordy, coordn, coord, Lcheb,
                    Lfour, yresps-1, xresps-1, rank, numprocs, &eventflag, iterations
                    *deltaT2BE, iterations*deltaT2BE*ylen/U, MPI_complex);
                printtheta2(theta1k, theta2k, &outputcountps, outputsperfile, outstr,                1198
                     outputoption, Lcheb, Lfour,yresps-1, xresps-1, rank,  numprocs,
                    &eventflag, iterations*deltaT2BE, iterations*deltaT2BE*ylen/U,
                    pbackps, inps, outps, pchebbackps, inchebps, outchebps,
                    MPI_complex);}
#endif                                                                                               1199
                                                                                                     1200
// Print output done ============================================================                    1201
                                                                                                     1202
#ifdef PAS_SCAL                                                                                      1203
                if((iterations > deltaTps/deltaT2BE *1.0/1.0 || iterations ==                        1204
                    deltaTps/deltaT2BE *1.0/1.0) && copycount == 0)
                        { copycomplex(theta1k, theta1knmin11,xresps*yresps);                         1205
                         copycomplex(theta2k, theta2knmin11,xresps*yresps);                          1206
                        if(rank == 0)printf("\nnmin1 copied at iteration: %lu\n",                    1207
                            iterations);
```

```
                        copycount++;                                                    1208
                        }                                                               1209
                if((iterations > deltaTps/deltaTuv -1 || iterations == deltaTps/        1210
                    deltaTuv -1) && copycount2 == 0)
                        { copycomplex(psik, psiknmin11,xres*yres);                      1211
                         copycomplex(omegak, omegaknmin11,xres*yres);                   1212
                        if(rank == 0)printf("\nnmin1 for uv copied at iteration: %lu\   1213
                            n",iterations);
                        copycount2++;                                                   1214
                        }                                                               1215
#endif                                                                                  1216
                                                                                        1217
//printf("\n%d# 3\n",rank);                                                             1218
                                                                                        1219
#ifdef POINCARE                                                                         1220
 if((int)(iterations/2) != iterations/2.0)                                              1221
        { getchebu1kcol2Dc (psik/*nmin1*/, ukint, Lcheb, yres-1, xres-1);//changed      1222
         getfouru1krow2Dc (psik/*nmin1*/, vkint, Lfour, yres-1, xres-1);//changed       1223
         for(i = 0; i<(yres)*(xres); i++)                                               1224
                {vkint[i][0] = (-1.0)*vkint[i][0]; vkint[i][1] = (-1.0)*vkint[i][1];}   1225
        }                                                                               1226
 else                                                                                   1227
        {                                                                               1228
         getchebu1kcol2Dc (psik, uk, Lcheb, yres-1, xres-1);                            1229
         getfouru1krow2Dc (psik, vk, Lfour, yres-1, xres-1);                            1230
         for(i = 0; i<(yres)*(xres); i++)                                               1231
                {vk[i][0] = (-1.0)*vk[i][0]; vk[i][1] = (-1.0)*vk[i][1];}               1232
                                                                                        1233
        tracker1(coordpcare, coordnpcare, uk, ukint, ukn, vk, vkint, vkn, deltaT2BE     1234
            *2, coordnumpcare, Lfour,rank, numprocs, yres-1, xres-1, MPI_complex);
                                                                                        1235
        copycomplex(uk, ukn,(yres)*(xres));                                             1236
        copycomplex(vk, vkn,(yres)*(xres));                                             1237
                                                                                        1238
        for(i = 0; i<coordnumpcare*2;i++)                                               1239
        {coordnpcare[i] = coordpcare[i];}                                               1240
                                                                                        1241
        if(fabs(iterations*deltaT2BE - ((int)(coordfinalcountpcare)) * switchtime*2)    1242
            <1e-12 || iterations*deltaT2BE > ((int)(coordfinalcountpcare)) *
            switchtime*2)
                {copyarray(coordpcare , &(coordfinalpcare[coordnumpcare*2*((int)(       1243
                    coordfinalcountpcare))]), coordnumpcare*2);coordfinalcountpcare =
                     coordfinalcountpcare+1.0;}
                                                                                        1244
        if(fabs(iterations*deltaT2BE - coordouttimepcare * coordoutcountpcare)<1e-12    1245
            || iterations*deltaT2BE > coordouttimepcare * coordoutcountpcare)
                {printcoord(coordfinalpcare, &coordoutcountpcare,outputsperfile,        1246
                    outstr, outputoption, ((int)(time*(ylen/U)+1)) * coordnumpcare,
                    rank, iterations*deltaT2BE, iterations*deltaT2BE*ylen/U);
                        if(rank == 0)                                                   1247
                          {end = clock();                                               1248
                          dif = ((double) (end - start)) / CLOCKS_PER_SEC;              1249
                          printf("\n P'care file no: %g,  TIME TAKEN = %gs, time = %g   1250
                              , dim time = %gs, coordfinalcountpcare = %g\n",
                              coordoutcountpcare-1, dif, iterations*deltaT2BE,
                              iterations*deltaT2BE*ylen/U,coordfinalcountpcare);
                         }                                                              1251
                }                                                                       1252
        }                                                                               1253
                                                                                        1254
#endif                                                                                  1255
                                                                                        1256
#ifdef TRACKER                                                                          1257
 if((int)(iterations/2) != iterations/2.0)                                              1258
```

```
        { getchebu1kcol2Dc (psik/*nmin1*/, ukint, Lcheb, yres-1, xres-1);//changed
         getfouru1krow2Dc (psik/*nmin1*/, vkint, Lfour, yres-1, xres-1);//changed
         for(i = 0; i<(yres)*(xres); i++)
                {vkint[i][0] = (-1.0)*vkint[i][0]; vkint[i][1] = (-1.0)*vkint[i][1];}
        }
 else
        {
         getchebu1kcol2Dc (psik, uk, Lcheb, yres-1, xres-1);
         getfouru1krow2Dc (psik, vk, Lfour, yres-1, xres-1);
         for(i = 0; i<(yres)*(xres); i++)
                {vk[i][0] = (-1.0)*vk[i][0]; vk[i][1] = (-1.0)*vk[i][1];}

        tracker1(coordpcare, coordnpcare, uk, ukint, ukn, vk, vkint, vkn, deltaT2BE
            *2, coordnumpcare, Lfour,rank, numprocs, yres-1, xres-1, MPI_complex);

        copycomplex(uk, ukn,(yres)*(xres));
        copycomplex(vk, vkn,(yres)*(xres));

        for(i = 0; i<coordnumpcare*2;i++)
        {coordnpcare[i] = coordpcare[i];}

//      if(iterations*deltaT2BE == coordfinalcountpcare * switchtime*2 || iterations*
    deltaT2BE > coordfinalcountpcare * switchtime*2)
//              {copyarray(coordpcare , &(coordfinalpcare[coordnumpcare*2*((int)(
    coordfinalcountpcare) + 1)]), coordnumpcare*2);}

        if(fabs(iterations*deltaT2BE - coordouttimepcare * coordoutcountpcare)<1e-12
            || iterations*deltaT2BE > coordouttimepcare * coordoutcountpcare)
                {printcoord(coordpcare, &coordoutcountpcare, outstr, outputsperfile,
                    outputoption, coordnumpcare, rank, iterations*deltaT2BE,
                    iterations*deltaT2BE*ylen/U);
                     if(rank == 0)
                       {end = clock();
                       dif = ((double) (end - start)) / CLOCKS_PER_SEC;
                       printf("\n Tracker file no: %g,  TIME TAKEN = %gs, time = %
                           g, dim time = %gs\n",coordoutcountpcare-1, dif,
                           iterations*deltaT2BE, iterations*deltaT2BE*ylen/U);
                      }
                }
        }

#endif

        }


        }

if(rank == 0)printf("\nIterations for 2nd order Backward Euler done! TIME = %g\n",
    iterations*deltaT2BE);
//Iterations for 2nd order Backward Euler done!================================
// ============================================================================
#ifdef PAS_SCAL
         copycomplex(theta1knmin11, theta1knmin1,xresps*yresps);
         copycomplex(theta2knmin11, theta2knmin1,xresps*yresps);
         copycomplex(psiknmin11, psiknmin1,xres*yres);
         copycomplex(psiknmin11, omegaknmin1,xres*yres);


          if(rank == 0){maxc(psik,yres*xres); printf("\n#%d: For theta1: ",rank);maxc
              (theta1k,xresps*yresps);}

        #ifdef DEBUG
         if(rank == 0)
```

```
        {printuv(psik, in, out, pback,incheb, outcheb, pchebback, Lcheb,Lfour,yres
            -1, xres -1);
         chebbackcol2Dc(psik,temp2D1,incheb,outcheb,pchebback,yres -1, xres -1,
             MPI_complex);
         fourbackrow2Dc(temp2D1,temp2D2,in,out,pback,yres -1, xres -1, MPI_complex);
         printf("\npsi:\n");print2Dc(temp2D2,yres,xres);

         chebbackcol2Dc(omegak,temp2D1,incheb,outcheb,pchebback,yres -1, xres -1,
             MPI_complex);
         fourbackrow2Dc(temp2D1,temp2D2,in,out,pback,yres -1, xres -1, MPI_complex);
         printf("\nomega:\n");print2Dc(temp2D2,yres,xres);

         printf("\npsik:\n");print2Dc(psik,yres,xres);
         printf("\nomegak:\n");print2Dc(omegak,yres,xres);

         getfouru1krow2Dc (omegak, temp2D2,Lfour,yres-1,  xres-1);
         printf("\nomegak(1):\n");print2Dc(temp2D2,yres,xres);
         chebbackcol2Dc(temp2D2,temp2D1,incheb,outcheb,pchebback,yres -1, xres -1,
             MPI_complex);
         fourbackrow2Dc(temp2D1,temp2D2,in,out,pback,yres -1, xres -1, MPI_complex);
         printf("\nomega(1):\n");print2Dc(temp2D2,yres,xres);
        }
        #endif
        if(rank == 0)
                {if (itconv ==0)
                        printf("\nConvergence not achieved.\n");
                else
                        if(rank == 0) printf("\n Convergence achieved at iteration =
                            %lu & time = %g\n",itconv, deltaT2BE*itconv*(ylen/U));
                }

// Assemble Influence Matrix for 4th order Backward Euler=======================

            calc2BEomegak12(omegak1, boundaryval1, Re, deltaTuv,in,out,pback,pfor,
                incheb,outcheb,pchebback,Lcheb,Lfour, yres-1,xres-1,n1, n2, rank,
                tasks, tasksnum, numtasks,numprocs, MPI_complex);
            calc2BEomegak12(omegak2, boundaryval2, Re, deltaTuv,in,out,pback,pfor,
                incheb,outcheb,pchebback,Lcheb,Lfour, yres-1,xres-1,n1, n2, rank,
                tasks, tasksnum, numtasks,numprocs, MPI_complex);
            calcpsik(psik1, omegak1,Lcheb, Lfour, yres-1, xres-1,n1, n2, rank,tasks,
                tasksnum, numtasks,numprocs, MPI_complex);
            calcpsik(psik2, omegak2,Lcheb, Lfour, yres-1, xres-1,n1, n2, rank,tasks,
                tasksnum, numtasks,numprocs, MPI_complex);
// Set boundaryvalt
        if(boundtime > 0.0)
                for(i = 0;i<xres;i++)
                {boundaryvalt[i] = (boundaryval[i+xres] - boundaryval[i])
                    /2.0*sin(iterations*deltaTuv*2.0*PI/boundtime);
                boundaryvalt[i+xres] = (-1.0) * boundaryvalt[i];
                }

            asmbinfmat(Ainv, psik1, psik2,  in, out, pback, incheb, outcheb,
                pchebback,boundaryvalt, Lcheb,yres-1,xres-1,n1, n2, rank,tasks,
                tasksnum, numtasks,numprocs, MPI_complex);

        #ifdef DEBUG
        if(rank == 0)
                {chebbackcol2Dc(psik1,temp2D1,incheb,outcheb,pchebback,yres-1,xres-1)
                    ;
                printf("\nFour coeffs of psik1:\n");print2Dc(temp2D1,yres,xres);
                chebbackcol2Dc(psik2,temp2D1,incheb,outcheb,pchebback,yres-1,xres-1);
                printf("\nFour coeffs of psik2:\n");print2Dc(temp2D1,yres,xres);
             printf("\nAinv: \n"); print2Dc(Ainv,xres,4);
```

```
                }                                                                                    1362
        #endif                                                                                       1363
                                                                                                     1364
// end asmb influence matrix ==================================================                      1365
                                                                                                     1366
        // printf("ITERATION: %d\n",iterations);                                                     1367
                                                                                                     1368
         if(rank == 0)                                                                               1369
                {if (itconv ==0)                                                                     1370
                        printf("\nConvergence not achieved.\n");                                     1371
                else                                                                                 1372
                 if(rank ==0)printf("\n Convergence achieved at iteration = %lu &                    1373
                     time = %g\n",itconv, deltaT2BE*itconv*(ylen/U));
                }                                                                                    1374
        //              printuv(psik, in, out, pback,incheb, outcheb, pchebback,                     1375
            Lcheb,Lfour,yres -1, xres -1);////////////////////////////////////
                                                                                                     1376
         if(rank == 0){maxc(psik,yres*xres);printf("\nFor theta1: ");maxc(theta1k,                   1377
             xresps*yresps);}
        // fprintuv(output2fp,psikn, in, out, pback,incheb, outcheb, pchebback, Lcheb                1378
            ,Lfour,yres -1, xres -1);
                                                                                                     1379
                                                                                                     1380
                                                                                                     1381
        if(time == 0.0)                                                                              1382
                iter = ITER;                                                                         1383
        else                                                                                         1384
                iter = (int)(time/deltaTuv);                                                         1385
iterations = (int)(2.0/1.0*deltaTps/deltaTuv) + 1;                                                   1386
//iterations = 3;deltaTuv                                                                            1387
//Iterations for 4th order Backward Euler ====================================                     1388
// ============================================================================                      1389
        if(rank == 0)printf("\nIterations for 4th order Backward Euler started! TIME                 1390
            = %g\n",iterations*deltaTuv);
                                                                                                     1391
        if (flag);                                                                                   1392
        else                                                                                         1393
        {for (iterations = (int)(2.0/1.0*deltaTps/deltaTuv) + 1,itconv = 0;iterations                1394
             <iter+1; iterations++)
                {if(rank==0)if(iterations/20==iterations/20.0)                                       1395
                  {end = clock();                                                                    1396
                  dif = ((double) (end - start)) / CLOCKS_PER_SEC;                                   1397
                  printf("\niteration: %lu\t\tprocess# %d: TIME TAKEN = %gs \t\t Non                 1398
                      dim time = %g \t\t dim time = %gs\n",iterations,rank,dif,
                      iterations*deltaTuv, iterations*deltaTuv*(ylen/U));
                  printf("\nFor theta1: ");maxc(theta1k,xresps*yresps);                             1399
                }                                                                                    1400
// Set boundaryvalt                                                                                  1401
        if(boundtime > 0.0)                                                                          1402
                {for(i = 0;i<xres;i++)                                                               1403
                        {boundaryvalt[i] = (boundaryval[i+xres] - boundaryval[i])                    1404
                            /2.0*sin(iterations*deltaTuv*2.0*PI/boundtime);
                        boundaryvalt[i+xres] = (-1.0) * boundaryvalt[i];                             1405
                        }                                                                            1406
                 asmbinfmat(Ainv, psik1, psik2,  in, out, pback, incheb, outcheb,                    1407
                     pchebback,boundaryvalt, Lcheb,yres-1,xres-1,n1, n2, rank,tasks,
                     tasksnum, numtasks,numprocs, MPI_complex);
                }                                                                                    1408
                                                                                                     1409
        if(reversetime > 0.0)                                                                        1410
        if(((iterations-1)*deltaTuv == reversetime || (iterations-1)*deltaTuv >                      1411
            reversetime) && reversecount == 0)
                {for(i = 0;i<xres*2;i++)                                                             1412
                                {boundaryvalt[i] = -boundaryvalt[i];                                 1413
```

```
                                boundaryvalop[i] = -boundaryvalop[i];                   1414
                                boundaryval[i] = -boundaryval[i];                       1415
                                }                                                       1416
                                                                                        1417
                 asmbinfmat(Ainv, psik1, psik2,  in, out, pback, incheb, outcheb,       1418
                     pchebback,boundaryvalt, Lcheb,yres-1,xres-1,n1, n2, rank,tasks,
                     tasksnum, numtasks,numprocs, MPI_complex);
                switchcount++;itconv =0;  eventflag = 1;                                1419
                if(rank == 0) printf("\nBoundary Velocities reversed at t = %16.16g [   1420
                    s]\n",iterations*deltaTuv*(ylen/U));
                reversecount++;                                                         1421
                }                                                                       1422
                                                                                        1423
        if(switchtime > 0.0)                                                            1424
        if((iterations-1)*deltaTuv == switchcount*switchtime || (iterations-1)*         1425
            deltaTuv > switchcount*switchtime)
                {if(phase == 0)                                                         1426
                        {phase = 1;                                                     1427
                        for(i = 0;i<xres*2;i++)                                         1428
                                {boundaryvalt[i] = boundaryvalop[i];                    1429
                                }                                                       1430
                        }                                                               1431
                else                                                                    1432
                        {phase = 0;                                                     1433
                        for(i = 0;i<xres*2;i++)                                         1434
                                {boundaryvalt[i] = boundaryval[i];                      1435
                                }                                                       1436
                        }                                                               1437
                 asmbinfmat(Ainv, psik1, psik2,  in, out, pback, incheb, outcheb,       1438
                     pchebback,boundaryvalt, Lcheb,yres-1,xres-1,n1, n2, rank,tasks,
                     tasksnum, numtasks,numprocs, MPI_complex);
                switchcount++;itconv =0;  eventflag = 1;                                1439
                if(rank == 0) printf("\nBoundary Velocities switched at t = %16.16g [   1440
                    s]\n",iterations*deltaTuv*(ylen/U));
                }                                                                       1441
                                                                                        1442
                                                                                        1443
// Calculate uv ================================================================        1444
//                    calc2BEomegaktilde(omegaktilde,  psikn,psiknmin1, omegakn,       1445
    omegaknmin1, in, out, pfor, pback, incheb, outcheb, pchebfor, pchebback, indeal,
    outdeal, pfordeal, pbackdeal, inchebdeal, outchebdeal, pchebfordeal,
    pchebbackdeal, deltaTuv, Re, Lcheb, Lfour, yresdeal-1, xresdeal-1, yres-1, xres
    -1,n1, n2, rank,tasks, tasksnum, numtasks,numprocs, MPI_complex);
                        calc2BEomegaktilde(omegaktilde, psikn, omegakn, omegan,         1446
                            omeganmin1, NLn, NLnmin1, psiknmin1, omegaknmin1, in, out
                            , pfor, pback, incheb, outcheb, pchebfor, pchebback,
                            indeal, outdeal, pfordeal, pbackdeal, inchebdeal,
                            outchebdeal, pchebfordeal, pchebbackdeal, deltaTuv, Re,
                            Lcheb, Lfour, yresdeal-1, xresdeal-1, yres-1, xres-1, n1,
                             n2, rank, tasks, tasksnum, numtasks, numprocs,
                            MPI_complex);
                         calcpsik(psiktilde, omegaktilde,Lcheb, Lfour, yres-1,xres-1,   1447
                             n1, n2, rank,tasks, tasksnum, numtasks,numprocs,
                             MPI_complex);
                         calclambda(lambda, psiktilde, Ainv,in, out, pfor,pback,        1448
                             incheb, outcheb, pchebback, boundaryvalt, Lcheb,yres-1,
                             xres-1,n1, n2, rank,tasks, tasksnum, numtasks,numprocs,
                             MPI_complex);
                 calcomegak(omegak, omegaktilde, omegak1, omegak2, lambda, incheb,      1449
                     outcheb,pchebfor,pchebback,yres -1, xres -1,n1, n2, rank,tasks,
                     tasksnum, numtasks,numprocs, MPI_complex);
                 calcomegak(psik, psiktilde,psik1, psik2, lambda, incheb,outcheb,       1450
                     pchebfor,pchebback,yres-1, xres-1,n1, n2, rank,tasks, tasksnum,
                     numtasks,numprocs, MPI_complex);
```

```
//printf("\n iteration: %lu, Time = %g\n",iterations,deltaTuv*iterations);
                        if(itconv == 0)
                        if(checkconvc(psik,psikn,ERROR,xres*yres))
                                {itconv = iterations;
                                if(time ==0.0)
                                        flag = 1.0;
                                else
                                        if(rank == 0) printf("\n Convergence achieved
                                             at iteration = %lu & time = %gs, nondim
                                            time = %g\n",iterations, deltaTuv*itconv
                                            *(ylen/U),deltaTuv*itconv);
                                }

                 copycomplex(omegakn, omegaknmin1,xres*yres);
                 copycomplex(psikn, psiknmin1,xres*yres);
                 copycomplex(omegak, omegakn,xres*yres);
                 copycomplex(psik, psikn,xres*yres);
// ========================================================================
                if(iterations*deltaTuv == hscount*hs/2 || iterations*deltaTuv>hscount
                    *hs/2)
                {calcuvRK4(psik, uRK4, uRK4i, vRK4, vRK4i, Lcheb, Lfour, rk4ratio
                    *2*3+1, yres-1, xres-1);
                 hscount++;
                /*if(rank == 0)
                {printf("\n++++++++++++++++++++++++\nTIME = %g\n",iterations*deltaTuv
                    );
                for(i = 0;i<rk4ratio *2*3+1;i++)
                        {printf("\n uk%d:\n",i);
                        print2D(uRK4i[i],yres,xres);
                        }
                printf("\npsik:\n");
                print2Dc(psik,yres,xres);
                }*/
//              printf("\n uRK4 written\n");
                }

                if(iterations*deltaTuv == rk4count*deltaTps || iterations*deltaTuv >
                    rk4count*deltaTps)
                        {
                        if(rank_theta1!=-1)
                        calcps3BE(theta1k, uRK4, uRK4i, vRK4, vRK4i, theta1kn,
                            theta1knmin1, theta1knmin2,theta1k2nmin1, theta1k2nmin2,
                            rk4ratio, rk4ratio *2*3+1, inps, outps, pforps, pbackps,
                            inchebps, outchebps, pchebforps, pchebbackps, deltaTps,
                            indealps, outdealps, pfordealps, pbackdealps,
                            inchebdealps, outchebdealps, pchebfordealps,
                            pchebbackdealps, Re, Sc1, Lcheb, Lfour, xres,yres,
                            yresdealps -1 , xresdealps - 1, yresps -1 , xresps - 1,
                            n1ps1, n2ps1, rank_theta1, tasksps1, tasksnumps1,
                            numtasksps1, procallocnum_theta[0], comm_theta1,
                            MPI_complex);
                if(rank_theta2 != -1)
                if(theta2flag ==1)
                        calcps3BE(theta2k, uRK4, uRK4i, vRK4, vRK4i, theta2kn,
                            theta2knmin1, theta2knmin2,theta2k2nmin1, theta2k2nmin2,
                            rk4ratio, rk4ratio *2*3+1, inps, outps, pforps, pbackps,
                            inchebps, outchebps, pchebforps, pchebbackps, indealps,
                            outdealps, pfordealps, pbackdealps, inchebdealps,
                            outchebdealps, pchebfordealps, pchebbackdealps,  deltaTps
                            , Re, Sc2, Lcheb, Lfour, xres,yres, yresdealps -1 ,
                            xresdealps - 1, yresps -1 , xresps - 1, n1ps2, n2ps2,
                            rank_theta2, tasksps2, tasksnumps2, numtasksps2,
                            procallocnum_theta[1], comm_theta2, MPI_complex);
```

```
                copycomplex(theta1knmin1, theta1knmin2,xresps*yresps);
                copycomplex(theta2knmin1, theta2knmin2,xresps*yresps);
                copycomplex(theta1kn, theta1knmin1,xresps*yresps);
                copycomplex(theta2kn, theta2knmin1,xresps*yresps);
                copycomplex(theta1k, theta1kn,xresps*yresps);
                copycomplex(theta2k, theta2kn,xresps*yresps);
                        rk4count++;
//      printf("\nPS calculated\n");
                    }
//printf("\n%d# 1\n",rank);
        #ifdef DEBUG
        if(rank == 0)
                {       chebbackcol2Dc(omegaktilde,temp2D1,incheb,outcheb,pchebback,
                    yres-1,xres-1, MPI_complex);
                        printf("\nFourier Coeffs of Omegaktilde: \n");print2Dc(
                            temp2D1,yres,xres);

                chebbackcol2Dc(psiktilde,temp2D1,incheb,outcheb,pchebback,yres-1,xres
                    -1, MPI_complex);
                        printf("\nFourier Coeffs of psiktilde: \n");print2Dc(temp2D1,
                            yres,xres);
                printf("\nLambda:%d\n",iterations);print2Dc(lambda,2,xres);
                        printf("\nomegaktilde\n"); print2Dc(omegaktilde,yres,xres);
                        printf("\nomegak\n"); print2Dc(omegak,yres,xres);

                        printf("\npsiktilde\n"); print2Dc(psiktilde,yres,xres);
                        printf("\npsik\n"); print2Dc(psik,yres,xres);

                        printf("\n\nIteration: %d\n\n",iterations);
                }
        #endif



if(flag) break;


        #ifdef DEBUG
        if(rank == 0)
                        {printf("\n!!!!!!!!!!!!!!!!!!!!!!!!\n");
                        printuv(psik, in, out, pback,incheb, outcheb, pchebback,
                            Lcheb,Lfour,yres -1, xres -1)
                            ;////////////////////////////////////
                        }
        #endif
        if(rank == 0);

//Prints out output periodically=================================================
        if(fabs(iterations*deltaTuv - outputcount*outtime)<1e-10 || iterations*
            deltaTuv > outputcount*outtime )
                printpsiuv(psik, &outputcount, outputsperfile, outstr, outputoption,
                    valoutput, coordx, coordy, coordn, coord, Lcheb, Lfour, yres-1,
                    xres-1, rank, numprocs, &eventflag, iterations*deltaTuv,
                    iterations*deltaTuv*ylen/U, MPI_complex);
        if(fabs(iterations*deltaTuv - outputcountps*outtimeps)<1e-10 || iterations*
            deltaTuv > outputcountps*outtimeps )
                {//printf("\nprocalloc_theta[1] = %d\n",procalloc_theta[1]);
                                if (rank == procalloc_theta[1])
                                {MPI_Send(theta2k, yresps*xresps, MPI_complex, 0,
                                    150,MPI_COMM_WORLD);}
                                else if (rank == 0)
                                {MPI_Recv(theta2k, yresps*xresps, MPI_complex,
                                    procalloc_theta[1], 150, MPI_COMM_WORLD,&Stat);}
```

```
                //printtheta(theta1k, theta2k, &outputcountps, outputsperfile, outstr
                    , outputoption, valoutput, coordx, coordy, coordn, coord, Lcheb,
                    Lfour, yresps-1, xresps-1, rank, numprocs, &eventflag, iterations
                    *deltaTuv, iterations*deltaTuv*ylen/U, MPI_complex);
                printtheta2(theta1k, theta2k, &outputcountps, outputsperfile, outstr,
                     outputoption, Lcheb, Lfour,yresps-1, xresps-1, rank,  numprocs,
                    &eventflag, iterations*deltaTuv, iterations*deltaTuv*ylen/U,
                    pbackps, inps, outps, pchebbackps, inchebps, outchebps,
                    MPI_complex);
                }
// Print output done ============================================================

//printf("\n%d# 3\n",rank);
        }


        }
//Iterations for 4th order Backward Euler done ================================
// ============================================================================

        // printf("ITERATION: %d\n",iterations);

         if(rank == 0)
                {if (itconv ==0)
                        printf("\nConvergence not achieved.\n");
                else
                 if(rank ==0)printf("\n Convergence achieved at iteration = %lu &
                     time = %g\n",itconv, deltaTuv*itconv*(ylen/U));
                }
        //              printuv(psik, in, out, pback,incheb, outcheb, pchebback,
            Lcheb,Lfour,yres -1, xres -1);////////////////////////////////////

         if(rank == 0){maxc(psik,yres*xres);printf("\nFor theta1: ");maxc(theta1k,
              xresps*yresps);}
        // fprintuv(output2fp,psikn, in, out, pback,incheb, outcheb, pchebback, Lcheb
            ,Lfour,yres -1, xres -1);
#endif

        #ifdef DEBUG
         if(rank == 0)
        { printf("\n\nBoundary Velocities:\n");
         print2D(boundaryval,2,xres);
        }
        #endif

        //print2D(coord,coordy,coordx*2);
         if(rank == 0) printf("\n\n");

        // for(i = 0;i<coordx;i++)
        //      printf("  %g",coord[(2*coordx)+2*i]);
        // printf("\n\n");

        MPI_Finalize();

if(rank == 0)
        {
        option = 'y';
        if(option == 'y' || option == 'Y')
        {outputmaxfp = fopen(outputfile,"w");


         fprintf(outputmaxfp,"\nRe : %g\nTIME : %g\nSIZE : %d  %d\nLENGTHS : %g  %g\
             nUMAX :  %g\n",Re,time,xres,yres,xlen,ylen,U);
```

```
         fprintf(outputmaxfp,"\npsik = \n"); fprint2Dc2(outputmaxfp,psik,yres,xres);
         getfouru1krow2Dc (psik, dpsidxk, Lfour,yres-1, xres-1);
         fprintf(outputmaxfp,"\ndpsidxk = \n"); fprint2Dc2(outputmaxfp,dpsidxk,yres,
             xres);
         getfouru1krow2Dc (dpsidxk, d2psidx2k, Lfour,yres-1, xres-1);
         fprintf(outputmaxfp,"\nd2psidx2k = \n"); fprint2Dc2(outputmaxfp,d2psidx2k,
             yres,xres);
         getchebu1kcol2Dc (dpsidxk, d2psidxdyk, Lcheb, yres-1, xres-1);
         fprintf(outputmaxfp,"\nd2psidxdyk = \n"); fprint2Dc2(outputmaxfp,d2psidxdyk,
             yres,xres);
         getchebu1kcol2Dc (psik, dpsidyk, Lcheb, yres-1, xres-1);
         fprintf(outputmaxfp,"\ndpsidyk = \n"); fprint2Dc2(outputmaxfp,dpsidyk,yres,
             xres);
         getchebu1kcol2Dc (dpsidyk, d2psidy2k, Lcheb, yres-1, xres-1);
         fprintf(outputmaxfp,"\nd2psidy2k = \n"); fprint2Dc2(outputmaxfp,d2psidy2k,
             yres,xres);

        }
        fclose(outputmaxfp);
        }

  end = clock();
  dif = ((double) (end - start)) / CLOCKS_PER_SEC;
printf("\nprocess# %d: TIME TAKEN = %gs\n",rank,dif);

}
    return 1;
}
```

High level sub-routines

```

#include<fftw3.h>
#include<stdio.h>
#include<stdlib.h>
#include<stddef.h>
#include<ctype.h>
#include<math.h>
#include<mpi.h>
#include<string.h>

#define STRMAX 50
//#define SIZE 15
#define PI 3.141592653589793
//#define ITER 6000
//#define DEBUG
/*
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Assembles the K matrix for evaluating Omegak for 1st order Backward Euler, with
    Cheb columnwise and Four rowwise
// Note K is assembled for one column at a time
// Temp note: Real Part = Imag Part
int asmbKBE(fftw_complex *K, double Re, double deltaT,int colno,double Lcheb, double
    Lfour, int m,int n)
{int l,k,p,multiplier;
 double ck;
 double sigma;
if(colno<ceil1(n+1,2)) multiplier = colno;else multiplier = (colno-n-1);
sigma = -(1.0*multiplier*multiplier*(2.0*PI/Lfour)*(2.0*PI/Lfour) + Re/deltaT);
 makezeroc(K, (m+1)*(m+1));
 for(l = 0;l<m-2+1;l++)
         for(p = l+2;p<m+1;p++)
             if ((p+l)/2 == (p+l)/2.0)
                 {if (l==0)ck=2.0; else ck = 1.0; K[(l+2)*(m+1)+p][0] +=(2.0/Lcheb)
```

```
                    *(2.0/Lcheb)*p*(p*p-l*l)/ck; K[(l+2)*(m+1)+p][1] +=(2.0/Lcheb)
                    *(2.0/Lcheb)*p*(p*p-l*l)/ck;}
 for(l = 0; l<m-2+1;l++)
      {K[(l+2)*(m+1)+l][0]+= sigma; K[(l+2)*(m+1)+l][1]+= sigma;}

 for(p = 0;p<m+1;p++)
      {K[(0)*(m+1) +p][0]+=pow(-1,p)*1.0; K[(0)*(m+1) +p][1]+=pow(-1,p)*1.0;
       K[1*(m+1)+p][0]+=1.0; K[1*(m+1)+p][1]+=1.0;
      }
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*/


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Assembles the K matrix for evaluating Omegak for 1st order Backward Euler, with
    Cheb columnwise and Four rowwise
// Note K is assembled for one column at a time
// Temp note: Real Part = Imag Part
int asmbKBE(fftw_complex *K, double Re, double deltaT,int colno,double Lcheb, double
    Lfour, int m,int n)
{int l,k,p,multiplier;
 double ck;
 double sigma;
if(colno<ceil1(n+1,2)) multiplier = colno;else multiplier = (colno-n-1);
sigma = -(1.0*multiplier*multiplier*(2.0*PI/Lfour)*(2.0*PI/Lfour) + Re/deltaT);
 makezeroc(K, (m+1)*(m+1));
 for(l = 0;l<m-2+1;l++)
         for(p = l+2;p<m+1;p++)
             if ((p+l)/2 == (p+l)/2.0)
                 {if (l==0)ck=2.0; else ck = 1.0; K[l*(m+1)+p][0] +=(2.0/Lcheb)*(2.0/
                     Lcheb)*p*(p*p-l*l)/ck; K[l*(m+1)+p][1] +=(2.0/Lcheb)*(2.0/Lcheb)
                     *p*(p*p-l*l)/ck;}

 for(l = 0; l<m-2+1;l++)
      {K[l*(m+1)+l][0]+= sigma; K[l*(m+1)+l][1]+= sigma;}

 for(p = 0;p<m+1;p++)
      {K[(m-1)*(m+1) +p][0]+=pow(-1,p)*1.0; K[(m-1)*(m+1) +p][1]+=pow(-1,p)*1.0;
       K[m*(m+1)+p][0]+=1.0; K[m*(m+1)+p][1]+=1.0;
      }
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Assembles the K matrix for evaluating Omegak for 2nd order Backward Euler, with
    Cheb columnwise and Four rowwise
// Note K is assembled for one column at a time
// Temp note: Real Part = Imag Part
int asmbK2BE(fftw_complex *K, double Re, double deltaT,int colno,double Lcheb, double
     Lfour, int m,int n)
{int l,k,p,multiplier;
 double ck;
 double sigma;
if(colno<ceil1(n+1,2)) multiplier = colno;else multiplier = (colno-n-1);
 sigma = -(1.0*multiplier*multiplier*(2.0*PI/Lfour)*(2.0*PI/Lfour) + 1.5*Re/deltaT);
 makezeroc(K, (m+1)*(m+1));
 for(l = 0;l<m-2+1;l++)
         for(p = l+2;p<m+1;p++)
             if ((p+l)/2 == (p+l)/2.0)
                 {if (l==0)ck=2.0; else ck = 1.0; K[l*(m+1)+p][0] +=(2.0/Lcheb)*(2.0/
```

```
                    Lcheb)*p*(p*p-l*l)/ck; K[l*(m+1)+p][1] +=(2.0/Lcheb)*(2.0/Lcheb)
                    *p*(p*p-l*l)/ck;}
 for(l = 0; l<m-2+1;l++)
      {K[l*(m+1)+l][0]+= sigma; K[l*(m+1)+l][1]+= sigma;}

 for(p = 0;p<m+1;p++)
      {K[(m-1)*(m+1) +p][0]+=pow(-1,p)*1.0; K[(m-1)*(m+1) +p][1]+=pow(-1,p)*1.0;
       K[m*(m+1)+p][0]+=1.0; K[m*(m+1)+p][1]+=1.0;
      }
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Assembles the K matrix for evaluating Omegak for 1st order Backward Euler, with
    Cheb columnwise and Four rowwise
// Note K is assembled for one column at a time
// Temp note: Real Part = Imag Part
int asmbKBEred(fftw_complex *Kodd, fftw_complex *Keven, double Re, double deltaT,int
    colno,double Lcheb, double Lfour, int m,int n)
{int l,k,p,multiplier;
 int meven, modd;
 double ck;
 double sigma;
 double Pk, Qk, Rk;
 int leven, lodd;
 if((m+1)/2.0 != (int)((m+1)/2.0 ))
        {meven = ceil1(m+1,2); modd = (m+1)/2;}
 else
        {modd = ceil1(m+1,2); meven = (m+1)/2;}

if(colno<ceil1(n+1,2)) multiplier = colno;else multiplier = (colno-n-1);
 sigma = -(1.0*multiplier*multiplier*(2.0*PI/Lfour)*(2.0*PI/Lfour) + Re/deltaT);
 makezeroc(Keven, (meven)*(meven));
 makezeroc(Kodd, (modd)*(modd));
 leven = 1;
 lodd = 1;
 for(l = 2;l<m+1;l++)
         {if(l/2.0 == (int)(l/2.0))
                        {if (l==2) Pk = 2.0/(4.0*l*(l-1));
                        else  Pk = 1.0/(4.0*l*(l-1));
                        if (l<m+1-2) Qk = (-1.0)/(2.0*(l*l-1));
                        else Qk = 0.0;
                        if (l<m+1-4) Rk = (1.0)/(4.0*l*(l+1));
                        else Rk = 0.0;
                        Keven[leven*(meven)+leven-1][0] = Keven[leven*(meven)+leven
                            -1][1] =  Pk*sigma;
                        Keven[leven*(meven)+leven][0] = Keven[leven*(meven)+leven][1]
                             =  Qk*sigma+(2.0/Lcheb)*(2.0/Lcheb)*1.0;
                        if(l<m+1-2)
                        Keven[leven*(meven)+leven+1][0] = Keven[leven*(meven)+leven
                            +1][1] =  Rk*sigma;
                        leven++;
                        }
                else
                        {if (l==2) Pk = 2.0/(4.0*l*(l-1));
                        else  Pk = 1.0/(4.0*l*(l-1));
                        if (l<m+1-2) Qk = (-1.0)/(2.0*(l*l-1));
                        else Qk = 0.0;
                        if (l<m+1-4) Rk = (1.0)/(4.0*l*(l+1));
                        else Rk = 0.0;
                        Kodd[lodd*(modd)+lodd-1][0] = Kodd[lodd*(modd)+lodd-1][1] =
                            Pk*sigma;
```

```
                        Kodd[lodd*(modd)+lodd][0] = Kodd[lodd*(modd)+lodd][1] =  Qk*
                            sigma+(2.0/Lcheb)*(2.0/Lcheb)*1.0;
                        if(l<m+1-2)
                        Kodd[lodd*(modd)+lodd+1][0] = Kodd[lodd*(modd)+lodd+1][1] =
                            Rk*sigma;
                        lodd++;
                        }
                }

 for(p = 0;p<meven;p++)
      {Keven[p][0]= 1.0; Keven[p][1]=1.0;}

 for(p = 0;p<modd;p++)
      {Kodd[p][0]= 1.0; Kodd[p][1]=1.0;}

 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Assembles the K matrix for evaluating Omegak for 2nd order Backward Euler, with
    Cheb columnwise and Four rowwise
// Note K is assembled for one column at a time
// Temp note: Real Part = Imag Part
int asmbK2BEred(fftw_complex *Kodd, fftw_complex *Keven, double Re, double deltaT,int
     colno,double Lcheb, double Lfour, int m,int n)
{int l,k,p,multiplier;
 int meven, modd;
 double ck;
 double sigma;
 double Pk, Qk, Rk;
 int leven, lodd;
 if((m+1)/2.0 != (int)((m+1)/2.0 ))
        {meven = ceil1(m+1,2); modd = (m+1)/2;}
 else
        {modd = ceil1(m+1,2); meven = (m+1)/2;}

if(colno<ceil1(n+1,2)) multiplier = colno;else multiplier = (colno-n-1);
 sigma = -(1.0*multiplier*multiplier*(2.0*PI/Lfour)*(2.0*PI/Lfour) + 1.5*Re/deltaT);
 makezeroc(Keven, (meven)*(meven));
 makezeroc(Kodd, (modd)*(modd));
 leven = 1;
 lodd = 1;
 for(l = 2;l<m+1;l++)
         {if(l/2.0 == (int)(l/2.0))
                        {if (l==2) Pk = 2.0/(4.0*l*(l-1));
                        else  Pk = 1.0/(4.0*l*(l-1));
                        if (l<m+1-2) Qk = (-1.0)/(2.0*(l*l-1));
                        else Qk = 0.0;
                        if (l<m+1-4) Rk = (1.0)/(4.0*l*(l+1));
                        else Rk = 0.0;
                        Keven[leven*(meven)+leven-1][0] = Keven[leven*(meven)+leven
                            -1][1] =  Pk*sigma;
                        Keven[leven*(meven)+leven][0] = Keven[leven*(meven)+leven][1]
                             =  Qk*sigma+1.0;
                        if(l<m+1-2)
                        Keven[leven*(meven)+leven+1][0] = Keven[leven*(meven)+leven
                            +1][1] =  Rk*sigma;
                        leven++;
                        }
                else
                        {if (l==2) Pk = 2.0/(4.0*l*(l-1));
```

```
                        else  Pk = 1.0/(4.0*l*(l-1));
                        if (l<m+1-2) Qk = (-1.0)/(2.0*(l*l-1));
                        else Qk = 0.0;
                        if (l<m+1-4) Rk = (1.0)/(4.0*l*(l+1));
                        else Rk = 0.0;
                        Kodd[lodd*(modd)+lodd-1][0] = Kodd[lodd*(modd)+lodd-1][1] =
                            Pk*sigma;
                        Kodd[lodd*(modd)+lodd][0] = Kodd[lodd*(modd)+lodd][1] =  Qk*
                            sigma+1.0;
                        if(l<m+1-2)
                        Kodd[lodd*(modd)+lodd+1][0] = Kodd[lodd*(modd)+lodd+1][1] =
                            Rk*sigma;
                        lodd++;
                        }
                }

 for(p = 0;p<meven;p++)
      {Keven[p][0]= 1.0; Keven[p][1]=1.0;}

 for(p = 0;p<modd;p++)
      {Kodd[p][0]= 1.0; Kodd[p][1]=1.0;}

 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Assembles the K matrix for evaluating Omegak for 2nd order Backward Euler, with
    Cheb columnwise and Four rowwise
// Note K is assembled for one column at a time
// Temp note: Real Part = Imag Part
int asmbK3BE(fftw_complex *K, double Re, double deltaT,int colno,double Lcheb, double
     Lfour, int m,int n)
{int l,k,p,multiplier;
 double ck;
 double sigma;
if(colno<ceil1(n+1,2)) multiplier = colno;else multiplier = (colno-n-1);
 sigma = -(1.0*multiplier*multiplier*(2.0*PI/Lfour)*(2.0*PI/Lfour) + 11.0/6.0*Re/
     deltaT);
 makezeroc(K, (m+1)*(m+1));
 for(l = 0;l<m-2+1;l++)
         for(p = l+2;p<m+1;p++)
             if ((p+l)/2 == (p+l)/2.0)
                 {if (l==0)ck=2.0; else ck = 1.0; K[l*(m+1)+p][0] +=(2.0/Lcheb)*(2.0/
                     Lcheb)*p*(p*p-l*l)/ck; K[l*(m+1)+p][1] +=(2.0/Lcheb)*(2.0/Lcheb)
                     *p*(p*p-l*l)/ck;}

 for(l = 0; l<m-2+1;l++)
      {K[l*(m+1)+l][0]+= sigma; K[l*(m+1)+l][1]+= sigma;}

 for(p = 0;p<m+1;p++)
      {K[(m-1)*(m+1) +p][0]+=pow(-1,p)*1.0; K[(m-1)*(m+1) +p][1]+=pow(-1,p)*1.0;
       K[m*(m+1)+p][0]+=1.0; K[m*(m+1)+p][1]+=1.0;
      }
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Assembles the K matrix for evaluating Omegak for 2nd order Backward Euler, with
    Cheb columnwise and Four rowwise
// Note K is assembled for one column at a time
// Temp note: Real Part = Imag Part
int asmbK4BE(fftw_complex *K, double Re, double deltaT,int colno,double Lcheb, double
```

```
    Lfour, int m,int n)
{int l,k,p,multiplier;
 double ck;
 double sigma;
if(colno<ceil1(n+1,2)) multiplier = colno;else multiplier = (colno-n-1);
 sigma = -(1.0*multiplier*multiplier*(2.0*PI/Lfour)*(2.0*PI/Lfour) + (25.0/12.0)*Re/
    deltaT);
 makezeroc(K, (m+1)*(m+1));
 for(l = 0;l<m-2+1;l++)
         for(p = l+2;p<m+1;p++)
             if ((p+l)/2 == (p+l)/2.0)
                 {if (l==0)ck=2.0; else ck = 1.0; K[l*(m+1)+p][0] +=(2.0/Lcheb)*(2.0/
                     Lcheb)*p*(p*p-l*l)/ck; K[l*(m+1)+p][1] +=(2.0/Lcheb)*(2.0/Lcheb)
                     *p*(p*p-l*l)/ck;}

 for(l = 0; l<m-2+1;l++)
      {K[l*(m+1)+l][0]+= sigma; K[l*(m+1)+l][1]+= sigma;}

 for(p = 0;p<m+1;p++)
      {K[(m-1)*(m+1) +p][0]+=pow(-1,p)*1.0; K[(m-1)*(m+1) +p][1]+=pow(-1,p)*1.0;
       K[m*(m+1)+p][0]+=1.0; K[m*(m+1)+p][1]+=1.0;
      }
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Assembles the F-vector for 1st order BE
int assembleFBE(fftw_complex *F, fftw_complex *psik, fftw_complex *omegak,
    fftw_complex *in, fftw_complex *out, fftw_plan pfor, fftw_plan pback,
    fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebfor, fftw_plan
    pchebback, fftw_complex *indeal, fftw_complex *outdeal, fftw_plan pfordeal,
    fftw_plan pbackdeal, fftw_complex *inchebdeal, fftw_complex *outchebdeal,
    fftw_plan pchebfordeal, fftw_plan pchebbackdeal, double deltaT, double Re, double
     Lcheb, double Lfour, int mdeal, int ndeal, int m, int n, MPI_Datatype
    MPI_complex)

{int i,j,k,colno,rowno;
 fftw_complex *uBE, *domegakBE, *domegaBE, *temp1BE, *temp2BE, *temp3BE, *FtempBE;
#ifdef DEBUG
 {printf("\nassembleFBE\n");}//////////////////
#endif
 uBE = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 domegakBE = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 domegaBE = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp1BE = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp2BE = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp3BE = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 FtempBE = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));

 //===========assemble (u*dwdx)k=========================

 changeres(psik, temp2BE, m+1, n+1, mdeal+1, ndeal+1);
 changeres(omegak, temp3BE, m+1, n+1, mdeal+1, ndeal+1);

 calcupar(temp2BE, uBE, indeal, outdeal, pbackdeal,inchebdeal, outchebdeal,
    pchebbackdeal,Lcheb, mdeal, ndeal, MPI_complex);
 getfouru1krow2Dc (temp3BE, domegakBE,Lfour, mdeal,  ndeal);

 chebbackcol2Dcpar(domegakBE, temp1BE, inchebdeal, outchebdeal, pchebbackdeal, mdeal
    ,ndeal, MPI_complex);
 fourbackrow2Dcpar(temp1BE, domegaBE, indeal, outdeal, pbackdeal, mdeal, ndeal,
    MPI_complex);
```

```
 dotmultcrim(uBE,domegaBE,FtempBE,(mdeal+1)*(ndeal+1));
#ifdef DEBUG
{ printf("\ndomegadx\n"); print2Dc(domegaBE,m+1,n+1);
 printf("\nu\n");print2Dc(uBE,m+1,n+1);
 printf("\nu*domegadx\n"); print2Dc(FtempBE,m+1,n+1);}
#endif
/////================ assemble (v*dwdy)k======================
 calcvpar(temp2BE, uBE, indeal, outdeal, pbackdeal, inchebdeal, outchebdeal,
     pchebbackdeal, Lfour, mdeal, ndeal, MPI_complex);
 getchebu1kcol2Dc (temp3BE, domegakBE, Lcheb, mdeal, ndeal);
// dealiasecol2D(domegakBE, m, n);//changed
 chebbackcol2Dcpar(domegakBE, temp1BE, inchebdeal, outchebdeal, pchebbackdeal,mdeal,
     ndeal, MPI_complex);
 fourbackrow2Dcpar(temp1BE, domegaBE, indeal, outdeal, pbackdeal, mdeal, ndeal,
     MPI_complex);

 dotmultcrim(uBE,domegaBE,temp1BE,(mdeal+1)*(ndeal+1));
#ifdef DEBUG
{ printf("\ndomegady\n"); print2Dc(domegaBE,m+1,n+1);
 printf("\nv\n");print2Dc(uBE,m+1,n+1);
 printf("\nv*domegady\n"); print2Dc(temp1BE,m+1,n+1);}
#endif

 addcomplex(temp1BE,FtempBE,(mdeal+1)*(ndeal+1));
#ifdef DEBUG
{ printf("\nFtempBE: (should be real only)\n");print2Dc(FtempBE,m+1,n+1);}
#endif
 chebbackcol2Dcpar(temp3BE,temp1BE,inchebdeal,outchebdeal,pchebbackdeal,mdeal,ndeal,
     MPI_complex);
 fourbackrow2Dcpar(temp1BE,temp2BE,indeal,outdeal,pbackdeal,mdeal,ndeal, MPI_complex)
     ;

////////==============subtract wk/deltT====================

 for(i = 0;i<(mdeal+1)*(ndeal+1);i++)
 {FtempBE[i][0]=Re*(FtempBE[i][0] - temp2BE[i][0]/deltaT);  FtempBE[i][1]= Re*(
     FtempBE[i][1] - temp2BE[i][1]/deltaT);}
 chebforcol2Dcpar(FtempBE,temp1BE,inchebdeal,outchebdeal,pchebfordeal,mdeal,ndeal,
     MPI_complex);
 fourforrow2Dcpar(temp1BE,temp2BE,indeal,outdeal,pfordeal,mdeal,ndeal, MPI_complex);
/*
 changeres(temp2BE, F, mdeal+1, ndeal+1, m+1, n+1);
//===================Setting Boundary Condition = 0.0==================
 for(colno = 0;colno<n+1;colno++)
    {F[(m-1)*(n+1)+colno][0] = F[(m-1)*(n+1)+colno][1] = 0.0;
     F[(m)*(n+1)+colno][0] = F[(m)*(n+1)+colno][1] = 0.0;
     }
*/
 changeres(temp2BE, F, mdeal+1, ndeal+1, m+1, n+1);
// changeres(temp2BE, temp1BE, mdeal+1, ndeal+1, m+1, n+1);
//===================Setting Boundary Condition = 0.0==================
/*
copycomplex(temp1BE, &(F[2*(n+1)]), (m-1)*(n+1));
 for(colno = 0;colno<n+1;colno++)
    {F[(0)*(n+1)+colno][0] = F[(0)*(n+1)+colno][1] = 0.0;
     F[(1)*(n+1)+colno][0] = F[(1)*(n+1)+colno][1] = 0.0;
     }
*/

#ifdef DEBUG
{ printf("\nF:\n"); print2Dc(F,m+1,n+1);}
#endif

 fftw_free(uBE);
```

```
 fftw_free(domegakBE);
 fftw_free(domegaBE);
 fftw_free(temp1BE);
 fftw_free(temp2BE);
 fftw_free(temp3BE);
 fftw_free(FtempBE);

 return 1;

}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Assembles the F-vector for 2nd order BE
int assembleF2BE2(fftw_complex *F, fftw_complex *psikn, fftw_complex *psiknmin1,
    fftw_complex *omegakn,fftw_complex *omegaknmin1, fftw_complex *in, fftw_complex *
    out, fftw_plan pfor, fftw_plan pback, fftw_complex *incheb, fftw_complex *outcheb
    , fftw_plan pchebfor, fftw_plan pchebback, fftw_complex *indeal, fftw_complex *
    outdeal, fftw_plan pfordeal, fftw_plan pbackdeal, fftw_complex *inchebdeal,
    fftw_complex *outchebdeal, fftw_plan pchebfordeal, fftw_plan pchebbackdeal,
    double deltaT, double Re, double Lcheb, double Lfour,  int mdeal, int ndeal, int
    m, int n, MPI_Datatype MPI_complex)
{int i,j,k,colno,rowno;
 fftw_complex *u, *v, *domegadk, *domegad, *prodn, *prodnmin1,*temp1, *temp2,*Ftemp,*
     omegan,*omeganmin1;
#ifdef DEBUG
{ printf("\nassembleFBE\n");}//////////////////
#endif
 u = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
// v = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 domegadk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 domegad = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 prodn = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 prodnmin1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 Ftemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 omegan = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 omeganmin1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));


 //Assemble (2V.delw)n ============================================================
 // u*dwdx
 changeres(psikn, temp1, m+1,n+1, mdeal+1, ndeal+1);
 calcupar(temp1, u, indeal, outdeal, pbackdeal,inchebdeal, outchebdeal, pchebbackdeal
     ,Lcheb, mdeal, ndeal, MPI_complex);//changed
 changeres(omegakn, temp1, m+1,n+1, mdeal+1, ndeal+1);
 getfouru1krow2Dc (temp1, domegadk,Lfour, mdeal,  ndeal);

 chebbackcol2Dcpar(domegadk, temp1, inchebdeal, outchebdeal, pchebbackdeal, mdeal,
     ndeal, MPI_complex);
 fourbackrow2Dcpar(temp1, domegad, indeal, outdeal, pbackdeal, mdeal, ndeal,
     MPI_complex);

 dotmultcrim(u,domegad,temp1,(mdeal+1)*(ndeal+1));
#ifdef DEBUG
{ printf("\ndomegadx\n"); print2Dc(domegad,m+1,n+1);
 printf("\nu\n");print2Dc(u,m+1,n+1);
 printf("\nu*domegadx\n"); print2Dc(temp1,m+1,n+1);}
#endif

// v*dwdy
 changeres(psikn, temp2, m+1,n+1, mdeal+1, ndeal+1);
 calcvpar(temp2, u, indeal, outdeal, pbackdeal, inchebdeal, outchebdeal,
```

```
     pchebbackdeal , Lfour , mdeal , ndeal , MPI_complex);//changed
 changeres(omegakn , temp2 , m+1,n+1, mdeal+1, ndeal+1);
 getchebu1kcol2Dc (temp2 , domegadk , Lcheb , mdeal , ndeal);
// dealiasecol2D(domegadk , m, n);//changed
 chebbackcol2Dcpar(domegadk , temp2 , inchebdeal , outchebdeal , pchebbackdeal , mdeal ,
     ndeal , MPI_complex);
 fourbackrow2Dcpar(temp2 , domegad , indeal , outdeal , pbackdeal , mdeal , ndeal ,
     MPI_complex);

 dotmultcrim(u,domegad ,temp2 ,(mdeal+1)*(ndeal+1));
#ifdef DEBUG
{ printf("\ndomegady\n"); print2Dc(domegad ,m+1,n+1);
 printf("\nv\n");print2Dc(u,m+1,n+1);
 printf("\nv*domegady\n"); print2Dc(temp2 ,m+1,n+1);}
#endif

 for(i = 0;i<mdeal+1;i++)
        for(colno = 0;colno<ndeal+1;colno++)
                {prodn[i*(ndeal+1)+colno][0] = temp1[i*(ndeal+1)+colno][0] + temp2[i
                    *(ndeal+1)+colno][0];
                 prodn[i*(ndeal+1)+colno][1] = temp1[i*(ndeal+1)+colno][1] + temp2[i
                     *(ndeal+1)+colno][1];
                }
//============================================================================


 //Assemble (2V.delw)nmin1 ===================================================
 // u*dwdx
 changeres(psiknmin1 , temp1 , m+1,n+1, mdeal+1, ndeal+1);
 calcupar(temp1 , u , indeal , outdeal , pbackdeal ,inchebdeal , outchebdeal , pchebbackdeal
     ,Lcheb , mdeal , ndeal , MPI_complex);//changed

 changeres(omegaknmin1 , temp1 , m+1,n+1, mdeal+1, ndeal+1);
 getfouru1krow2Dc (temp1 , domegadk ,Lfour , mdeal ,  ndeal);
 chebbackcol2Dcpar(domegadk , temp1 , inchebdeal , outchebdeal , pchebbackdeal ,mdeal ,
     ndeal , MPI_complex);
 fourbackrow2Dcpar(temp1 , domegad , indeal , outdeal , pbackdeal , mdeal , ndeal ,
     MPI_complex);

 dotmultcrim(u,domegad ,temp1 ,(mdeal+1)*(ndeal+1));
#ifdef DEBUG
{ printf("\ndomegadx\n"); print2Dc(domegad ,m+1,n+1);
 printf("\nu\n");print2Dc(u,m+1,n+1);
 printf("\nu*domegadx\n"); print2Dc(temp1 ,m+1,n+1);}
#endif

// v*dwdy
 changeres(psiknmin1 , temp2 , m+1,n+1, mdeal+1, ndeal+1);
 calcvpar(temp2 , u , indeal , outdeal , pbackdeal , inchebdeal , outchebdeal ,
     pchebbackdeal , Lfour , mdeal , ndeal , MPI_complex);//changed

 changeres(omegaknmin1 , temp2 , m+1,n+1, mdeal+1, ndeal+1);
 getchebu1kcol2Dc (temp2 , domegadk , Lcheb , mdeal , ndeal);
 chebbackcol2Dcpar(domegadk , temp2 , inchebdeal , outchebdeal , pchebbackdeal ,mdeal ,
     ndeal , MPI_complex);
 fourbackrow2Dcpar(temp2 , domegad , indeal , outdeal , pbackdeal , mdeal , ndeal ,
     MPI_complex);

 dotmultcrim(u,domegad ,temp2 ,(mdeal+1)*(ndeal+1));
#ifdef DEBUG
{ printf("\ndomegady\n"); print2Dc(domegad ,m+1,n+1);
 printf("\nv\n");print2Dc(u,m+1,n+1);
 printf("\nv*domegady\n"); print2Dc(temp2 ,m+1,n+1);}
#endif
```

```
 for(i = 0;i<mdeal+1;i++)
        for(colno = 0;colno<ndeal+1;colno++)
                {prodnmin1[i*(ndeal+1)+colno][0] = temp1[i*(ndeal+1)+colno][0] +
                    temp2[i*(ndeal+1)+colno][0];
                 prodnmin1[i*(ndeal+1)+colno][1] = temp1[i*(ndeal+1)+colno][1] +
                     temp2[i*(ndeal+1)+colno][1];
                }
//=============================================================================

 changeres(omegakn, temp1, m+1,n+1, mdeal+1, ndeal+1);
 chebbackcol2Dcpar(temp1, u, inchebdeal, outchebdeal, pchebbackdeal,mdeal,ndeal,
     MPI_complex);
 fourbackrow2Dcpar(u, omegan, indeal, outdeal, pbackdeal, mdeal, ndeal, MPI_complex);

 changeres(omegaknmin1, temp2, m+1,n+1, mdeal+1, ndeal+1);
 chebbackcol2Dcpar(temp2, u, inchebdeal, outchebdeal, pchebbackdeal,mdeal,ndeal,
     MPI_complex);
 fourbackrow2Dcpar(u, omeganmin1, indeal, outdeal, pbackdeal, mdeal, ndeal,
     MPI_complex);

 for(i = 0;i<mdeal+1;i++)
        for(colno = 0;colno<ndeal+1;colno++)
                {Ftemp[i*(ndeal+1)+colno][0] = (-0.5)*Re/deltaT*((4.0)*omegan[i*(
                    ndeal+1)+colno][0] - omeganmin1[i*(ndeal+1)+colno][0]) + Re*(2*
                    prodn[i*(ndeal+1)+colno][0] - prodnmin1[i*(ndeal+1)+colno][0]);
                Ftemp[i*(ndeal+1)+colno][1] = (-0.5)*Re/deltaT*((4.0)*omegan[i*(ndeal
                    +1)+colno][1] - omeganmin1[i*(ndeal+1)+colno][1]) + Re*(2*prodn[i
                    *(ndeal+1)+colno][1] - prodnmin1[i*(ndeal+1)+colno][1]);
                }
 chebforcol2Dcpar(Ftemp,temp1,inchebdeal,outchebdeal,pchebfordeal,mdeal,ndeal,
     MPI_complex);
 fourforrow2Dcpar(temp1,temp2,indeal,outdeal,pfordeal,mdeal,ndeal, MPI_complex);
 changeres(temp2, F, mdeal+1, ndeal+1, m+1, n+1);
// changeres(temp2, temp1, mdeal+1, ndeal+1, m+1, n+1);
//==================Setting Boundary Condition = 0.0==================

/*copycomplex(temp1, &(F[2*(n+1)]), (m-1)*(n+1));
 for(colno = 0;colno<n+1;colno++)
    {F[(0)*(n+1)+colno][0] = F[(0)*(n+1)+colno][1] = 0.0;
     F[(1)*(n+1)+colno][0] = F[(1)*(n+1)+colno][1] = 0.0;
     }
*/
#ifdef DEBUG
{ printf("\nF:\n"); print2Dc(F,m+1,n+1);}
#endif

 fftw_free(u);
// fftw_free(v);
 fftw_free(domegadk);
 fftw_free(domegad);
 fftw_free(prodn);
 fftw_free(prodnmin1);
 fftw_free(temp1);
 fftw_free(temp2);
 fftw_free(Ftemp);
 fftw_free(omegan);
 fftw_free(omeganmin1);

 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Assembles the F-vector for 2nd order BE
int initF2BE(fftw_complex *psikn, fftw_complex *omegakn,fftw_complex *omegan,
    fftw_complex *omeganmin1, fftw_complex *NLn, fftw_complex *NLnmin1, fftw_complex
    *in, fftw_complex *out, fftw_plan pfor, fftw_plan pback, fftw_complex *incheb,
    fftw_complex *outcheb, fftw_plan pchebfor, fftw_plan pchebback, fftw_complex *
    indeal, fftw_complex *outdeal, fftw_plan pfordeal, fftw_plan pbackdeal,
    fftw_complex *inchebdeal, fftw_complex *outchebdeal, fftw_plan pchebfordeal,
    fftw_plan pchebbackdeal, double deltaT, double Re, double Lcheb, double Lfour,
    int mdeal, int ndeal, int m, int n, MPI_Datatype MPI_complex)
{int i,j,k,colno,rowno;
 double alpha = (-1.0*Re)/deltaT; int size2 = (mdeal+1)*(ndeal+1)*2, size = (mdeal+1)
     *(ndeal+1), one = 1;
 fftw_complex *u, *v, *domegadk, *domegad,*temp1, *temp2,*temp3, *Ftemp, *psikndeal,
     *omegakndeal;


 static int *procarray;
 static MPI_Group groupuddxn, groupuddxn_1, groupuddxn_2, groupvddyn, groupvddyn_1,
     groupvddyn_2, groupomegan, worldgroup;
 static int rankuddxn = -1, rankuddxn_1 = -1,rankuddxn_2 = -1, rankvddyn = -1,
     rankvddyn_1 = -1, rankvddyn_2 = -1, rankomegan = -1;
 static int procalloc1[2], procallocnum1[2], procalloc2[5], procallocnum2[5];
 static MPI_Comm commuddxn, commuddxn_1, commuddxn_2;
 static MPI_Comm commvddyn, commvddyn_1, commvddyn_2;
 static MPI_Comm commomegan;


 static int commflag = 0;


 static int numprocs, rank;

 static MPI_Status Stat;

 MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 MPI_Comm_group(MPI_COMM_WORLD, &worldgroup);
 u = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
// v = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 domegadk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 domegad = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp3 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 Ftemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 omegakndeal = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 psikndeal = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));


 procarray = malloc(sizeof(int)*numprocs);
 for(i = 0;i<numprocs;i++) procarray[i] = i;


if(commflag == 0)
{
if(numprocs >4)
{// Set the communicators ============================
        retseglengths(procalloc2, procallocnum2, numprocs, 5);
        procalloc1[0] = procalloc2[0];  procalloc1[1] = procalloc2[2];

        procallocnum1[0] = procallocnum2[0]+procallocnum2[1];  procallocnum1[1] =
            procallocnum2[2]+procallocnum2[3];
```

```
                                                                                    575
        MPI_Group_incl(worldgroup, procallocnum2[0], &(procarray[procalloc2[0]]), &  576
            groupuddxn_1);
        MPI_Group_incl(worldgroup, procallocnum2[1], &(procarray[procalloc2[1]]), &  577
            groupuddxn_2);
        MPI_Group_incl(worldgroup, procallocnum2[2], &(procarray[procalloc2[2]]), &  578
            groupvddyn_1);
        MPI_Group_incl(worldgroup, procallocnum2[3], &(procarray[procalloc2[3]]), &  579
            groupvddyn_2);
                                                                                    580
        MPI_Group_incl(worldgroup, procallocnum2[4], &(procarray[procalloc2[4]]), &  581
            groupomegan);
                                                                                    582
                                                                                    583
        MPI_Group_incl(worldgroup, procallocnum1[0], &(procarray[procalloc1[0]]), &  584
            groupuddxn);
        MPI_Group_incl(worldgroup, procallocnum1[1], &(procarray[procalloc1[1]]), &  585
            groupvddyn);
                                                                                    586
                                                                                    587
        MPI_Comm_create(MPI_COMM_WORLD, groupuddxn_1, &commuddxn_1);                588
        MPI_Comm_create(MPI_COMM_WORLD, groupuddxn_2, &commuddxn_2);                589
        MPI_Comm_create(MPI_COMM_WORLD, groupvddyn_1, &commvddyn_1);                590
        MPI_Comm_create(MPI_COMM_WORLD, groupvddyn_2, &commvddyn_2);                591
                                                                                    592
        MPI_Comm_create(MPI_COMM_WORLD, groupomegan, &commomegan);                  593
                                                                                    594
                                                                                    595
        MPI_Comm_create(MPI_COMM_WORLD, groupuddxn, &commuddxn);                    596
        MPI_Comm_create(MPI_COMM_WORLD, groupvddyn, &commvddyn);                    597
                                                                                    598
                                                                                    599
         if(rank<procalloc2[0]+procallocnum2[0])                                    600
                {MPI_Comm_rank(commuddxn_1, &rankuddxn_1);}                         601
        else if(rank<procalloc2[1]+procallocnum2[1])                                602
                {MPI_Comm_rank(commuddxn_2, &rankuddxn_2);}                         603
        else if(rank<procalloc2[2]+procallocnum2[2])                                604
                {MPI_Comm_rank(commvddyn_1, &rankvddyn_1);}                         605
        else if(rank<procalloc2[3]+procallocnum2[3])                                606
                {MPI_Comm_rank(commvddyn_2, &rankvddyn_2);}                         607
                                                                                    608
        else if(rank<procalloc2[4]+procallocnum2[4])                                609
                {MPI_Comm_rank(commomegan, &rankomegan);}                           610
                                                                                    611
         if(rank<procalloc1[0]+procallocnum1[0])                                    612
                {MPI_Comm_rank(commuddxn, &rankuddxn);}                             613
        else if(rank<procalloc1[1]+procallocnum1[1])                                614
                {MPI_Comm_rank(commvddyn, &rankvddyn);}                             615
}                                                                                   616
 commflag++;                                                                        617
                                                                                    618
}                                                                                   619
 //Assemble (2V.delw)n ============================================================ 620
                                                                                    621
 changeres(psikn, psikndeal, m+1, n+1, mdeal+1, ndeal+1);                           622
 changeres(omegakn, omegakndeal, m+1, n+1, mdeal+1, ndeal+1);                       623
                                                                                    624
 // u*dwdx                                                                          625
if(rankuddxn != -1)                                                                 626
{                                                                                   627
 if(rankuddxn_1 != -1)                                                              628
         calcupar2(psikndeal, u, indeal, outdeal, pbackdeal,inchebdeal, outchebdeal, 629
             pchebbackdeal,Lcheb, mdeal, ndeal, commuddxn_1, procallocnum2[0],
             rankuddxn_1, MPI_complex);
```

```
 if(rankuddxn_2 != -1)
        {getfouru1krow2Dc (omegakndeal, domegadk,Lfour, mdeal,  ndeal);
         chebbackcol2Dcpar2(domegadk, temp1, inchebdeal, outchebdeal, pchebbackdeal,
             mdeal,ndeal, commuddxn_2, procallocnum2[1], rankuddxn_2, MPI_complex);
         fourbackrow2Dcpar2(temp1, domegad, indeal, outdeal, pbackdeal, mdeal, ndeal,
              commuddxn_2, procallocnum2[1], rankuddxn_2, MPI_complex);
        }

 MPI_Bcast(u,(mdeal+1)*(ndeal+1),MPI_complex,0,commuddxn);
 MPI_Bcast(domegad,(mdeal+1)*(ndeal+1),MPI_complex,procallocnum2[0],commuddxn);

 dotmultcrim(u,domegad,temp1,(mdeal+1)*(ndeal+1));
}

// v*dwdy
if(rankvddyn != -1)
{
 if(rankvddyn_1 != -1)
         calcvpar2(psikndeal, u, indeal, outdeal, pbackdeal, inchebdeal, outchebdeal,
              pchebbackdeal, Lfour, mdeal, ndeal, commvddyn_1, procallocnum2[2],
             rankvddyn_1, MPI_complex);

 if(rankvddyn_2 != -1)
        {getchebu1kcol2Dc (omegakndeal, domegadk, Lcheb, mdeal, ndeal);
        chebbackcol2Dcpar2(domegadk, temp2, inchebdeal, outchebdeal, pchebbackdeal,
            mdeal,ndeal, commvddyn_2, procallocnum2[3], rankvddyn_2, MPI_complex);
        fourbackrow2Dcpar2(temp2, domegad, indeal, outdeal, pbackdeal, mdeal, ndeal,
            commvddyn_2, procallocnum2[3], rankvddyn_2, MPI_complex);
        }

 MPI_Bcast(u,(mdeal+1)*(ndeal+1),MPI_complex,0,commvddyn);
 MPI_Bcast(domegad,(mdeal+1)*(ndeal+1),MPI_complex,procallocnum2[2],commvddyn);

 dotmultcrim(u,domegad,temp2,(mdeal+1)*(ndeal+1));
}



//=============================================================================
//MPI_Barrier(MPI_COMM_WORLD);

if(rankomegan != -1)
        {chebbackcol2Dcpar2(omegakndeal, temp3, inchebdeal, outchebdeal,
            pchebbackdeal,mdeal,ndeal, commomegan, procallocnum2[4], rankomegan,
            MPI_complex);
         fourbackrow2Dcpar2(temp3, omegan, indeal, outdeal, pbackdeal, mdeal, ndeal,
             commomegan, procallocnum2[4], rankomegan, MPI_complex);
        }


  MPI_Bcast(temp1,(mdeal+1)*(ndeal+1),MPI_complex,procalloc2[0],MPI_COMM_WORLD);
  MPI_Bcast(temp2,(mdeal+1)*(ndeal+1),MPI_complex,procalloc2[2],MPI_COMM_WORLD);
  MPI_Bcast(omegan,(mdeal+1)*(ndeal+1),MPI_complex,procalloc2[4],MPI_COMM_WORLD);


 for(i = 0;i<mdeal+1;i++)
        for(colno = 0;colno<ndeal+1;colno++)
                {NLn[i*(ndeal+1)+colno][0] = temp1[i*(ndeal+1)+colno][0] + temp2[i*(
                    ndeal+1)+colno][0];
                 NLn[i*(ndeal+1)+colno][1] = temp1[i*(ndeal+1)+colno][1] + temp2[i*(
                     ndeal+1)+colno][1];
                }
```

```
 for(i = 0;i<mdeal+1;i++)
        for(colno = 0;colno<ndeal+1;colno++)
                {Ftemp[i*(ndeal+1)+colno][0] = (-0.5)*Re/deltaT*((4.0)*omegan[i*(
                    ndeal+1)+colno][0] - omeganmin1[i*(ndeal+1)+colno][0]) + Re*(2*
                    NLn[i*(ndeal+1)+colno][0] - NLnmin1[i*(ndeal+1)+colno][0]);
                Ftemp[i*(ndeal+1)+colno][1] = (-0.5)*Re/deltaT*((4.0)*omegan[i*(ndeal
                    +1)+colno][1] - omeganmin1[i*(ndeal+1)+colno][1]) + Re*(2*NLn[i*(
                    ndeal+1)+colno][1] - NLnmin1[i*(ndeal+1)+colno][1]);
                }

copycomplex(omegan, omeganmin1, (mdeal+1)*(ndeal+1));
copycomplex(NLn, NLnmin1, (mdeal+1)*(ndeal+1));

 fftw_free(u);
// fftw_free(v);
 fftw_free(domegadk);
 fftw_free(domegad);
 fftw_free(temp1);
 fftw_free(temp2);
 fftw_free(temp3);
 fftw_free(Ftemp);
 fftw_free(psikndeal);
 fftw_free(omegakndeal);
 free(procarray);

 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Assembles the F-vector for 2nd order BE
int assembleF2BE(fftw_complex *F, fftw_complex *psikn, fftw_complex *omegakn,
    fftw_complex *omegan,fftw_complex *omeganmin1, fftw_complex *NLn, fftw_complex *
    NLnmin1, fftw_complex *in, fftw_complex *out, fftw_plan pfor, fftw_plan pback,
    fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebfor, fftw_plan
    pchebback, fftw_complex *indeal, fftw_complex *outdeal, fftw_plan pfordeal,
    fftw_plan pbackdeal, fftw_complex *inchebdeal, fftw_complex *outchebdeal,
    fftw_plan pchebfordeal, fftw_plan pchebbackdeal, double deltaT, double Re, double
     Lcheb, double Lfour,  int mdeal, int ndeal, int m, int n, MPI_Datatype
    MPI_complex)
{int i,j,k,colno,rowno;
 double alpha = (-1.0*Re)/deltaT; int size2 = (mdeal+1)*(ndeal+1)*2, size = (mdeal+1)
     *(ndeal+1), one = 1;
 fftw_complex *u, *v, *domegadk, *domegad,*temp1, *temp2,*temp3, *Ftemp, *psikndeal,
     *omegakndeal;


 static int *procarray;
 static MPI_Group groupuddxn, groupuddxn_1, groupuddxn_2, groupvddyn, groupvddyn_1,
     groupvddyn_2, groupomegan, worldgroup;
 static int rankuddxn = -1, rankuddxn_1 = -1,rankuddxn_2 = -1, rankvddyn = -1,
     rankvddyn_1 = -1, rankvddyn_2 = -1, rankomegan = -1;
 static int procalloc1[2], procallocnum1[2], procalloc2[5], procallocnum2[5];
 static MPI_Comm commuddxn, commuddxn_1, commuddxn_2;
 static MPI_Comm commvddyn, commvddyn_1, commvddyn_2;
 static MPI_Comm commomegan;


 static int commflag = 0;
```

```
 static int numprocs, rank;
 static MPI_Status Stat;
 MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 MPI_Comm_group(MPI_COMM_WORLD, &worldgroup);
 u = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
// v = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 domegadk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 domegad = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp3 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 Ftemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 omegakndeal = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 psikndeal = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));

 procarray = malloc(sizeof(int)*numprocs);
 for(i = 0;i<numprocs;i++) procarray[i] = i;

if(commflag == 0)
{
if(numprocs >4)
{// Set the communicators ============================
        retseglengths(procalloc2, procallocnum2, numprocs, 5);
        procalloc1[0] = procalloc2[0];  procalloc1[1] = procalloc2[2];

        procallocnum1[0] = procallocnum2[0]+procallocnum2[1];  procallocnum1[1] =
            procallocnum2[2]+procallocnum2[3];


        MPI_Group_incl(worldgroup, procallocnum2[0], &(procarray[procalloc2[0]]), &
            groupuddxn_1);
        MPI_Group_incl(worldgroup, procallocnum2[1], &(procarray[procalloc2[1]]), &
            groupuddxn_2);
        MPI_Group_incl(worldgroup, procallocnum2[2], &(procarray[procalloc2[2]]), &
            groupvddyn_1);
        MPI_Group_incl(worldgroup, procallocnum2[3], &(procarray[procalloc2[3]]), &
            groupvddyn_2);

        MPI_Group_incl(worldgroup, procallocnum2[4], &(procarray[procalloc2[4]]), &
            groupomegan);


        MPI_Group_incl(worldgroup, procallocnum1[0], &(procarray[procalloc1[0]]), &
            groupuddxn);
        MPI_Group_incl(worldgroup, procallocnum1[1], &(procarray[procalloc1[1]]), &
            groupvddyn);


        MPI_Comm_create(MPI_COMM_WORLD, groupuddxn_1, &commuddxn_1);
        MPI_Comm_create(MPI_COMM_WORLD, groupuddxn_2, &commuddxn_2);
        MPI_Comm_create(MPI_COMM_WORLD, groupvddyn_1, &commvddyn_1);
        MPI_Comm_create(MPI_COMM_WORLD, groupvddyn_2, &commvddyn_2);

        MPI_Comm_create(MPI_COMM_WORLD, groupomegan, &commomegan);


        MPI_Comm_create(MPI_COMM_WORLD, groupuddxn, &commuddxn);
        MPI_Comm_create(MPI_COMM_WORLD, groupvddyn, &commvddyn);
```

```
                                                                                                    787
                                                                                                    788
         if(rank<procalloc2[0]+procallocnum2[0])                                                    789
                {MPI_Comm_rank(commuddxn_1, &rankuddxn_1);}                                         790
        else if(rank<procalloc2[1]+procallocnum2[1])                                                791
                {MPI_Comm_rank(commuddxn_2, &rankuddxn_2);}                                         792
        else if(rank<procalloc2[2]+procallocnum2[2])                                                793
                {MPI_Comm_rank(commvddyn_1, &rankvddyn_1);}                                         794
        else if(rank<procalloc2[3]+procallocnum2[3])                                                795
                {MPI_Comm_rank(commvddyn_2, &rankvddyn_2);}                                         796
                                                                                                    797
        else if(rank<procalloc2[4]+procallocnum2[4])                                                798
                {MPI_Comm_rank(commomegan, &rankomegan);}                                           799
                                                                                                    800
         if(rank<procalloc1[0]+procallocnum1[0])                                                    801
                {MPI_Comm_rank(commuddxn, &rankuddxn);}                                             802
        else if(rank<procalloc1[1]+procallocnum1[1])                                                803
                {MPI_Comm_rank(commvddyn, &rankvddyn);}                                             804
}                                                                                                   805
 commflag++;                                                                                        806
                                                                                                    807
}                                                                                                   808
 //Assemble (2V.delw)n ============================================================               809
                                                                                                    810
 changeres(psikn, psikndeal, m+1, n+1, mdeal+1, ndeal+1);                                           811
 changeres(omegakn, omegakndeal, m+1, n+1, mdeal+1, ndeal+1);                                       812
                                                                                                    813
 // u*dwdx                                                                                          814
if(rankuddxn != -1)                                                                                 815
{                                                                                                   816
 if(rankuddxn_1 != -1)                                                                              817
         calcupar2(psikndeal, u, indeal, outdeal, pbackdeal,inchebdeal, outchebdeal,                818
             pchebbackdeal,Lcheb, mdeal, ndeal, commuddxn_1, procallocnum2[0],
             rankuddxn_1, MPI_complex);
                                                                                                    819
 if(rankuddxn_2 != -1)                                                                              820
        {getfouru1krow2Dc (omegakndeal, domegadk,Lfour, mdeal,  ndeal);                             821
         chebbackcol2Dcpar2(domegadk, temp1, inchebdeal, outchebdeal, pchebbackdeal,                822
             mdeal,ndeal, commuddxn_2, procallocnum2[1], rankuddxn_2, MPI_complex);
         fourbackrow2Dcpar2(temp1, domegad, indeal, outdeal, pbackdeal, mdeal, ndeal,               823
              commuddxn_2, procallocnum2[1], rankuddxn_2, MPI_complex);
        }                                                                                           824
                                                                                                    825
 MPI_Bcast(u,(mdeal+1)*(ndeal+1),MPI_complex,0,commuddxn);                                          826
 MPI_Bcast(domegad,(mdeal+1)*(ndeal+1),MPI_complex,procallocnum2[0],commuddxn);                     827
                                                                                                    828
 dotmultcrim(u,domegad,temp1,(mdeal+1)*(ndeal+1));                                                  829
}                                                                                                   830
                                                                                                    831
// v*dwdy                                                                                           832
if(rankvddyn != -1)                                                                                 833
{                                                                                                   834
 if(rankvddyn_1 != -1)                                                                              835
         calcvpar2(psikndeal, u, indeal, outdeal, pbackdeal, inchebdeal, outchebdeal,               836
              pchebbackdeal, Lfour, mdeal, ndeal, commvddyn_1, procallocnum2[2],
             rankvddyn_1, MPI_complex);
                                                                                                    837
 if(rankvddyn_2 != -1)                                                                              838
        {getchebu1kcol2Dc (omegakndeal, domegadk, Lcheb, mdeal, ndeal);                             839
        chebbackcol2Dcpar2(domegadk, temp2, inchebdeal, outchebdeal, pchebbackdeal,                 840
            mdeal,ndeal, commvddyn_2, procallocnum2[3], rankvddyn_2, MPI_complex);
        fourbackrow2Dcpar2(temp2, domegad, indeal, outdeal, pbackdeal, mdeal, ndeal,                841
            commvddyn_2, procallocnum2[3], rankvddyn_2, MPI_complex);
        }                                                                                           842
```

```
 MPI_Bcast(u,(mdeal+1)*(ndeal+1),MPI_complex,0,commvddyn);
 MPI_Bcast(domegad,(mdeal+1)*(ndeal+1),MPI_complex,procallocnum2[2],commvddyn);

 dotmultcrim(u,domegad,temp2,(mdeal+1)*(ndeal+1));
}



//============================================================================
//MPI_Barrier(MPI_COMM_WORLD);

if(rankomegan != -1)
        {chebbackcol2Dcpar2(omegakndeal, temp3, inchebdeal, outchebdeal,
            pchebbackdeal,mdeal,ndeal, commomegan, procallocnum2[4], rankomegan,
            MPI_complex);
         fourbackrow2Dcpar2(temp3, omegan, indeal, outdeal, pbackdeal, mdeal, ndeal,
             commomegan, procallocnum2[4], rankomegan, MPI_complex);
        }


  MPI_Bcast(temp1,(mdeal+1)*(ndeal+1),MPI_complex,procalloc2[0],MPI_COMM_WORLD);
  MPI_Bcast(temp2,(mdeal+1)*(ndeal+1),MPI_complex,procalloc2[2],MPI_COMM_WORLD);
  MPI_Bcast(omegan,(mdeal+1)*(ndeal+1),MPI_complex,procalloc2[4],MPI_COMM_WORLD);


 for(i = 0;i<mdeal+1;i++)
        for(colno = 0;colno<ndeal+1;colno++)
                {NLn[i*(ndeal+1)+colno][0] = temp1[i*(ndeal+1)+colno][0] + temp2[i*(
                    ndeal+1)+colno][0];
                 NLn[i*(ndeal+1)+colno][1] = temp1[i*(ndeal+1)+colno][1] + temp2[i*(
                     ndeal+1)+colno][1];
                }



 for(i = 0;i<mdeal+1;i++)
        for(colno = 0;colno<ndeal+1;colno++)
                {Ftemp[i*(ndeal+1)+colno][0] = (-0.5)*Re/deltaT*((4.0)*omegan[i*(
                    ndeal+1)+colno][0] - omeganmin1[i*(ndeal+1)+colno][0]) + Re*(2*
                    NLn[i*(ndeal+1)+colno][0] - NLnmin1[i*(ndeal+1)+colno][0]);
                Ftemp[i*(ndeal+1)+colno][1] = (-0.5)*Re/deltaT*((4.0)*omegan[i*(ndeal
                    +1)+colno][1] - omeganmin1[i*(ndeal+1)+colno][1]) + Re*(2*NLn[i*(
                    ndeal+1)+colno][1] - NLnmin1[i*(ndeal+1)+colno][1]);
                }

copycomplex(omegan, omeganmin1, (mdeal+1)*(ndeal+1));
copycomplex(NLn, NLnmin1, (mdeal+1)*(ndeal+1));

 chebforcol2Dcpar(Ftemp,temp1,inchebdeal,outchebdeal,pchebfordeal,mdeal,ndeal,
     MPI_complex);
 fourforrow2Dcpar(temp1,temp2,indeal,outdeal,pfordeal,mdeal,ndeal, MPI_complex);

 changeres(temp2, F, mdeal+1, ndeal+1, m+1, n+1);
/* changeres(temp2, temp1, mdeal+1, ndeal+1, m+1, n+1);
//====================Setting Boundary Condition = 0.0==================

copycomplex(temp1, &(F[2*(n+1)]), (m-1)*(n+1));
 for(colno = 0;colno<n+1;colno++)
    {F[(0)*(n+1)+colno][0] = F[(0)*(n+1)+colno][1] = 0.0;
     F[(1)*(n+1)+colno][0] = F[(1)*(n+1)+colno][1] = 0.0;
     }
*/
/* changeres(temp2, F, mdeal+1, ndeal+1, m+1, n+1);
```

```
//===================Setting Boundary Condition = 0.0==================
 for(colno = 0;colno<n+1;colno++)
    {F[(m-1)*(n+1)+colno][0] = F[(m-1)*(n+1)+colno][1] = 0.0;
     F[(m)*(n+1)+colno][0] = F[(m)*(n+1)+colno][1] = 0.0;
     }
*/
#ifdef DEBUG
{ printf("\nF:\n"); print2Dc(F,m+1,n+1);}
#endif

 fftw_free(u);
// fftw_free(v);
 fftw_free(domegadk);
 fftw_free(domegad);
 fftw_free(temp1);
 fftw_free(temp2);
 fftw_free(temp3);
 fftw_free(Ftemp);
 fftw_free(psikndeal);
 fftw_free(omegakndeal);
 free(procarray);

 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
init3FBE(fftw_complex *psikn, fftw_complex *omegakn,fftw_complex *omegan,
    fftw_complex *NLn, fftw_complex *in, fftw_complex *out, fftw_plan pfor, fftw_plan
     pback, fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebfor,
    fftw_plan pchebback, fftw_complex *indeal, fftw_complex *outdeal, fftw_plan
    pfordeal, fftw_plan pbackdeal, fftw_complex *inchebdeal, fftw_complex *
    outchebdeal, fftw_plan pchebfordeal, fftw_plan pchebbackdeal,  double deltaT,
    double Re,double Lcheb, double Lfour,  int mdeal, int ndeal, int m, int n,
    MPI_Datatype MPI_complex)
{int i,j,k,colno,rowno;
 double alpha = (-1.0*Re)/deltaT; int size2 = (mdeal+1)*(ndeal+1)*2, size = (mdeal+1)
     *(ndeal+1), one = 1;
 fftw_complex *u, *v, *domegadk, *domegad, *temp1, *temp2,*Ftemp,*temp3, *psikndeal,
     *omegakndeal;

 static int *procarray;
 static MPI_Group groupuddxn, groupuddxn_1, groupuddxn_2, groupvddyn, groupvddyn_1,
     groupvddyn_2, groupomegan, worldgroup;
 static int rankuddxn = -1, rankuddxn_1 = -1,rankuddxn_2 = -1, rankvddyn = -1,
     rankvddyn_1 = -1, rankvddyn_2 = -1, rankomegan = -1;
 static int procalloc1[2], procallocnum1[2], procalloc2[5], procallocnum2[5];
 static MPI_Comm commuddxn, commuddxn_1, commuddxn_2;
 static MPI_Comm commvddyn, commvddyn_1, commvddyn_2;
 static MPI_Comm commomegan;

 static int commflag = 0;

 static int numprocs, rank;
// static int blockcounts[1];
 static MPI_Status Stat;
/* static MPI_Datatype MPI_complex,oldtypes[1];
 static MPI_Aint offsets[1];
 offsets[0] = 0;
 blockcounts[0] = 2;
 oldtypes[0] = MPI_DOUBLE;
 MPI_Type_struct(1,blockcounts,offsets,oldtypes,&MPI_complex);
 MPI_Type_commit (&MPI_complex);
```

```
*/
 MPI_Comm_size(MPI_COMM_WORLD , &numprocs);
 MPI_Comm_rank(MPI_COMM_WORLD , &rank);
 MPI_Comm_group(MPI_COMM_WORLD , &worldgroup);

 u = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
// v = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 domegadk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 domegad = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp3 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 Ftemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 omegakndeal = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 psikndeal = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));

 procarray = malloc(sizeof(int)*numprocs);
 for(i = 0;i<numprocs;i++) procarray[i] = i;


if(commflag == 0)
{
if(numprocs >4)
{// Set the communicators ============================
        retseglengths(procalloc2, procallocnum2, numprocs, 5);
        procalloc1[0] = procalloc2[0];  procalloc1[1] = procalloc2[2];
        procallocnum1[0] = procallocnum2[0]+procallocnum2[1];  procallocnum1[1] =
            procallocnum2[2]+procallocnum2[3];


        MPI_Group_incl(worldgroup, procallocnum2[0], &(procarray[procalloc2[0]]), &
            groupuddxn_1);
        MPI_Group_incl(worldgroup, procallocnum2[1], &(procarray[procalloc2[1]]), &
            groupuddxn_2);
        MPI_Group_incl(worldgroup, procallocnum2[2], &(procarray[procalloc2[2]]), &
            groupvddyn_1);
        MPI_Group_incl(worldgroup, procallocnum2[3], &(procarray[procalloc2[3]]), &
            groupvddyn_2);

        MPI_Group_incl(worldgroup, procallocnum2[4], &(procarray[procalloc2[4]]), &
            groupomegan);


        MPI_Group_incl(worldgroup, procallocnum1[0], &(procarray[procalloc1[0]]), &
            groupuddxn);
        MPI_Group_incl(worldgroup, procallocnum1[1], &(procarray[procalloc1[1]]), &
            groupvddyn);


        MPI_Comm_create(MPI_COMM_WORLD, groupuddxn_1, &commuddxn_1);
        MPI_Comm_create(MPI_COMM_WORLD, groupuddxn_2, &commuddxn_2);
        MPI_Comm_create(MPI_COMM_WORLD, groupvddyn_1, &commvddyn_1);
        MPI_Comm_create(MPI_COMM_WORLD, groupvddyn_2, &commvddyn_2);

        MPI_Comm_create(MPI_COMM_WORLD, groupomegan, &commomegan);


        MPI_Comm_create(MPI_COMM_WORLD, groupuddxn, &commuddxn);
        MPI_Comm_create(MPI_COMM_WORLD, groupvddyn, &commvddyn);


         if(rank<procalloc2[0]+procallocnum2[0])
                {MPI_Comm_rank(commuddxn_1, &rankuddxn_1);}
        else if(rank<procalloc2[1]+procallocnum2[1])
```

```
                {MPI_Comm_rank(commuddxn_2, &rankuddxn_2);}
        else if(rank<procalloc2[2]+procallocnum2[2])
                {MPI_Comm_rank(commvddyn_1, &rankvddyn_1);}
        else if(rank<procalloc2[3]+procallocnum2[3])
                {MPI_Comm_rank(commvddyn_2, &rankvddyn_2);}

        else if(rank<procalloc2[4]+procallocnum2[4])
                {MPI_Comm_rank(commomegan, &rankomegan);}

         if(rank<procalloc1[0]+procallocnum1[0])
                {MPI_Comm_rank(commuddxn, &rankuddxn);}
        else if(rank<procalloc1[1]+procallocnum1[1])
                {MPI_Comm_rank(commvddyn, &rankvddyn);}
}
else rankuddxn = rankuddxn_1 = rankuddxn_2 = rankvddyn = rankvddyn_1 = rankvddyn_2 =
    rankomegan = rank;

 commflag++;

}
 //Assemble (2V.delw)n ============================================================

 changeres(psikn, psikndeal, m+1, n+1, mdeal+1, ndeal+1);
 changeres(omegakn, omegakndeal, m+1, n+1, mdeal+1, ndeal+1);

 // u*dwdx
if(rankuddxn != -1)
{
 if(rankuddxn_1 != -1)
         calcupar2(psikndeal, u, indeal, outdeal, pbackdeal,inchebdeal, outchebdeal,
             pchebbackdeal,Lcheb, mdeal, ndeal, commuddxn_1, procallocnum2[0],
             rankuddxn_1, MPI_complex);

 if(rankuddxn_2 != -1)
        {getfouru1krow2Dc (omegakndeal, domegadk,Lfour, mdeal,  ndeal);
         chebbackcol2Dcpar2(domegadk, temp1, inchebdeal, outchebdeal, pchebbackdeal,
             mdeal,ndeal, commuddxn_2, procallocnum2[1], rankuddxn_2, MPI_complex);
         fourbackrow2Dcpar2(temp1, domegad, indeal, outdeal, pbackdeal, mdeal, ndeal,
              commuddxn_2, procallocnum2[1], rankuddxn_2, MPI_complex);
        }

if(numprocs >4)
{ MPI_Bcast(u,(mdeal+1)*(ndeal+1),MPI_complex,0,commuddxn);
 MPI_Bcast(domegad,(mdeal+1)*(ndeal+1),MPI_complex,procallocnum2[0],commuddxn);
}
 dotmultcrim(u,domegad,temp1,(mdeal+1)*(ndeal+1));
}

// v*dwdy
if(rankvddyn != -1)
{
 if(rankvddyn_1 != -1)
         calcvpar2(psikndeal, u, indeal, outdeal, pbackdeal, inchebdeal, outchebdeal,
              pchebbackdeal, Lfour, mdeal, ndeal, commvddyn_1, procallocnum2[2],
             rankvddyn_1, MPI_complex);

 if(rankvddyn_2 != -1)
        {getchebu1kcol2Dc (omegakndeal, domegadk, Lcheb, mdeal, ndeal);
        chebbackcol2Dcpar2(domegadk, temp2, inchebdeal, outchebdeal, pchebbackdeal,
            mdeal,ndeal, commvddyn_2, procallocnum2[3], rankvddyn_2, MPI_complex);
        fourbackrow2Dcpar2(temp2, domegad, indeal, outdeal, pbackdeal, mdeal, ndeal,
            commvddyn_2, procallocnum2[3], rankvddyn_2, MPI_complex);
        }
if(numprocs >4)
```

```
{ MPI_Bcast(u,(mdeal+1)*(ndeal+1),MPI_complex,0,commvddyn);
 MPI_Bcast(domegad,(mdeal+1)*(ndeal+1),MPI_complex,procallocnum2[2],commvddyn);
}
 dotmultcrim(u,domegad,temp2,(mdeal+1)*(ndeal+1));
}


if(rankomegan != -1)
        {chebbackcol2Dcpar2(omegakndeal, temp3, inchebdeal, outchebdeal,
            pchebbackdeal,mdeal,ndeal, commomegan, procallocnum2[4], rankomegan,
            MPI_complex);
         fourbackrow2Dcpar2(temp3, omegan, indeal, outdeal, pbackdeal, mdeal, ndeal,
             commomegan, procallocnum2[4], rankomegan, MPI_complex);
        }


if(numprocs >4)
{  MPI_Bcast(temp1,(mdeal+1)*(ndeal+1),MPI_complex,procalloc2[0],MPI_COMM_WORLD);
  MPI_Bcast(temp2,(mdeal+1)*(ndeal+1),MPI_complex,procalloc2[2],MPI_COMM_WORLD);
  MPI_Bcast(omegan,(mdeal+1)*(ndeal+1),MPI_complex,procalloc2[4],MPI_COMM_WORLD);
}

 for(i = 0;i<mdeal+1;i++)
        for(colno = 0;colno<ndeal+1;colno++)
                {NLn[i*(ndeal+1)+colno][0] = temp1[i*(ndeal+1)+colno][0] + temp2[i*(
                    ndeal+1)+colno][0];
                 NLn[i*(ndeal+1)+colno][1] = temp1[i*(ndeal+1)+colno][1] + temp2[i*(
                     ndeal+1)+colno][1];
                }

#ifdef DEBUG
{ printf("\nF:\n"); print2Dc(F,m+1,n+1);}
#endif

 fftw_free(u);
// fftw_free(v);
 fftw_free(domegadk);
 fftw_free(domegad);
 fftw_free(temp1);
 fftw_free(temp2);
 fftw_free(temp3);
 fftw_free(Ftemp);
 fftw_free(psikndeal);
 fftw_free(omegakndeal);
 free(procarray);

 return 1;

}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Assembles the F-vector for 2nd order BE
int assembleF3BE(fftw_complex *F, fftw_complex *psikn, fftw_complex *omegakn,
    fftw_complex *omegan,fftw_complex *omeganmin1,fftw_complex *omeganmin2,
    fftw_complex *NLn, fftw_complex *NLnmin1, fftw_complex *NLnmin2, fftw_complex *in
    , fftw_complex *out, fftw_plan pfor, fftw_plan pback, fftw_complex *incheb,
    fftw_complex *outcheb, fftw_plan pchebfor, fftw_plan pchebback, fftw_complex *
    indeal, fftw_complex *outdeal, fftw_plan pfordeal, fftw_plan pbackdeal,
    fftw_complex *inchebdeal, fftw_complex *outchebdeal, fftw_plan pchebfordeal,
    fftw_plan pchebbackdeal, double deltaT, double Re, double Lcheb, double Lfour,
    int mdeal, int ndeal, int m, int n, MPI_Datatype MPI_complex)
{int i,j,k,colno,rowno;
 double alpha = (-1.0*Re)/deltaT; int size2 = (mdeal+1)*(ndeal+1)*2, size = (mdeal+1)
     *(ndeal+1), one = 1;
```

```
 fftw_complex *u, *v, *domegadk, *domegad, *temp1, *temp2,*Ftemp,*temp3, *psikndeal,
     *omegakndeal;

 static int *procarray;
 static MPI_Group groupuddxn, groupuddxn_1, groupuddxn_2, groupvddyn, groupvddyn_1,
     groupvddyn_2, groupomegan, worldgroup;
 static int rankuddxn = -1, rankuddxn_1 = -1,rankuddxn_2 = -1, rankvddyn = -1,
     rankvddyn_1 = -1, rankvddyn_2 = -1, rankomegan = -1;
 static int procalloc1[2], procallocnum1[2], procalloc2[5], procallocnum2[5];
 static MPI_Comm commuddxn, commuddxn_1, commuddxn_2;
 static MPI_Comm commvddyn, commvddyn_1, commvddyn_2;
 static MPI_Comm commomegan;

 static int commflag = 0;

 static int numprocs, rank;
// static int blockcounts[1];
 static MPI_Status Stat;
/* static MPI_Datatype MPI_complex,oldtypes[1];
 static MPI_Aint offsets[1];
 offsets[0] = 0;
 blockcounts[0] = 2;
 oldtypes[0] = MPI_DOUBLE;
 MPI_Type_struct(1,blockcounts,offsets,oldtypes,&MPI_complex);
 MPI_Type_commit (&MPI_complex);
*/
 MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 MPI_Comm_group(MPI_COMM_WORLD, &worldgroup);

 u = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
// v = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 domegadk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 domegad = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 temp3 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 Ftemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 omegakndeal = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));
 psikndeal = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(mdeal+1)*(ndeal+1));

 procarray = malloc(sizeof(int)*numprocs);
 for(i = 0;i<numprocs;i++) procarray[i] = i;

//printf("\n!!!!!!!!!!!!!!!!!!!!!\n");


if(commflag == 0)
{
if(numprocs >4)
{// Set the communicators ============================
        retseglengths(procalloc2, procallocnum2, numprocs, 5);
        procalloc1[0] = procalloc2[0];  procalloc1[1] = procalloc2[2];
        procallocnum1[0] = procallocnum2[0]+procallocnum2[1];  procallocnum1[1] =
            procallocnum2[2]+procallocnum2[3];


        MPI_Group_incl(worldgroup, procallocnum2[0], &(procarray[procalloc2[0]]), &
            groupuddxn_1);
        MPI_Group_incl(worldgroup, procallocnum2[1], &(procarray[procalloc2[1]]), &
            groupuddxn_2);
        MPI_Group_incl(worldgroup, procallocnum2[2], &(procarray[procalloc2[2]]), &
            groupvddyn_1);
        MPI_Group_incl(worldgroup, procallocnum2[3], &(procarray[procalloc2[3]]), &
```

```
            groupvddyn_2);
        MPI_Group_incl(worldgroup, procallocnum2[4], &(procarray[procalloc2[4]]), &
            groupomegan);

        MPI_Group_incl(worldgroup, procallocnum1[0], &(procarray[procalloc1[0]]), &
            groupuddxn);
        MPI_Group_incl(worldgroup, procallocnum1[1], &(procarray[procalloc1[1]]), &
            groupvddyn);

        MPI_Comm_create(MPI_COMM_WORLD, groupuddxn_1, &commuddxn_1);
        MPI_Comm_create(MPI_COMM_WORLD, groupuddxn_2, &commuddxn_2);
        MPI_Comm_create(MPI_COMM_WORLD, groupvddyn_1, &commvddyn_1);
        MPI_Comm_create(MPI_COMM_WORLD, groupvddyn_2, &commvddyn_2);

        MPI_Comm_create(MPI_COMM_WORLD, groupomegan, &commomegan);


        MPI_Comm_create(MPI_COMM_WORLD, groupuddxn, &commuddxn);
        MPI_Comm_create(MPI_COMM_WORLD, groupvddyn, &commvddyn);


         if(rank<procalloc2[0]+procallocnum2[0])
                {MPI_Comm_rank(commuddxn_1, &rankuddxn_1);}
        else if(rank<procalloc2[1]+procallocnum2[1])
                {MPI_Comm_rank(commuddxn_2, &rankuddxn_2);}
        else if(rank<procalloc2[2]+procallocnum2[2])
                {MPI_Comm_rank(commvddyn_1, &rankvddyn_1);}
        else if(rank<procalloc2[3]+procallocnum2[3])
                {MPI_Comm_rank(commvddyn_2, &rankvddyn_2);}

        else if(rank<procalloc2[4]+procallocnum2[4])
                {MPI_Comm_rank(commomegan, &rankomegan);}

         if(rank<procalloc1[0]+procallocnum1[0])
                {MPI_Comm_rank(commuddxn, &rankuddxn);}
        else if(rank<procalloc1[1]+procallocnum1[1])
                {MPI_Comm_rank(commvddyn, &rankvddyn);}
}
else rankuddxn = rankuddxn_1 = rankuddxn_2 = rankvddyn = rankvddyn_1 = rankvddyn_2 =
    rankomegan = rank;

 commflag++;

}
//printf("\n!!#################!!!\n");
 //Assemble (2V.delw)n ============================================================

 changeres(psikn, psikndeal, m+1, n+1, mdeal+1, ndeal+1);
 changeres(omegakn, omegakndeal, m+1, n+1, mdeal+1, ndeal+1);

 // u*dwdx
if(rankuddxn != -1)
{
 if(rankuddxn_1 != -1)
         calcupar2(psikndeal, u, indeal, outdeal, pbackdeal,inchebdeal, outchebdeal,
             pchebbackdeal,Lcheb, mdeal, ndeal, commuddxn_1, procallocnum2[0],
             rankuddxn_1, MPI_complex);

 if(rankuddxn_2 != -1)
        {getfouru1krow2Dc (omegakndeal, domegadk,Lfour, mdeal,  ndeal);
         chebbackcol2Dcpar2(domegadk, temp1, inchebdeal, outchebdeal, pchebbackdeal,
```

```
            mdeal,ndeal, commuddxn_2, procallocnum2[1], rankuddxn_2, MPI_complex);
         fourbackrow2Dcpar2(temp1, domegad, indeal, outdeal, pbackdeal, mdeal, ndeal,
              commuddxn_2, procallocnum2[1], rankuddxn_2, MPI_complex);
        }

if(numprocs >4)
{ MPI_Bcast(u,(mdeal+1)*(ndeal+1),MPI_complex,0,commuddxn);
 MPI_Bcast(domegad,(mdeal+1)*(ndeal+1),MPI_complex,procallocnum2[0],commuddxn);
}
 dotmultcrim(u,domegad,temp1,(mdeal+1)*(ndeal+1));
}

// v*dwdy
if(rankvddyn != -1)
{
 if(rankvddyn_1 != -1)
         calcvpar2(psikndeal, u, indeal, outdeal, pbackdeal, inchebdeal, outchebdeal,
              pchebbackdeal, Lfour, mdeal, ndeal, commvddyn_1, procallocnum2[2],
             rankvddyn_1, MPI_complex);

 if(rankvddyn_2 != -1)
        {getchebu1kcol2Dc (omegakndeal, domegadk, Lcheb, mdeal, ndeal);
        chebbackcol2Dcpar2(domegadk, temp2, inchebdeal, outchebdeal, pchebbackdeal,
            mdeal,ndeal, commvddyn_2, procallocnum2[3], rankvddyn_2, MPI_complex);
        fourbackrow2Dcpar2(temp2, domegad, indeal, outdeal, pbackdeal, mdeal, ndeal,
            commvddyn_2, procallocnum2[3], rankvddyn_2, MPI_complex);
        }
if(numprocs >4)
{ MPI_Bcast(u,(mdeal+1)*(ndeal+1),MPI_complex,0,commvddyn);
 MPI_Bcast(domegad,(mdeal+1)*(ndeal+1),MPI_complex,procallocnum2[2],commvddyn);
}
 dotmultcrim(u,domegad,temp2,(mdeal+1)*(ndeal+1));
}


if(rankomegan != -1)
        {chebbackcol2Dcpar2(omegakndeal, temp3, inchebdeal, outchebdeal,
            pchebbackdeal,mdeal,ndeal, commomegan, procallocnum2[4], rankomegan,
            MPI_complex);
         fourbackrow2Dcpar2(temp3, omegan, indeal, outdeal, pbackdeal, mdeal, ndeal,
            commomegan, procallocnum2[4], rankomegan, MPI_complex);
        }


if(numprocs >4)
{ MPI_Bcast(temp1,(mdeal+1)*(ndeal+1),MPI_complex,procalloc2[0],MPI_COMM_WORLD);
  MPI_Bcast(temp2,(mdeal+1)*(ndeal+1),MPI_complex,procalloc2[2],MPI_COMM_WORLD);
  MPI_Bcast(omegan,(mdeal+1)*(ndeal+1),MPI_complex,procalloc2[4],MPI_COMM_WORLD);
}

 for(i = 0;i<mdeal+1;i++)
        for(colno = 0;colno<ndeal+1;colno++)
                {NLn[i*(ndeal+1)+colno][0] = temp1[i*(ndeal+1)+colno][0] + temp2[i*(
                    ndeal+1)+colno][0];
                 NLn[i*(ndeal+1)+colno][1] = temp1[i*(ndeal+1)+colno][1] + temp2[i*(
                     ndeal+1)+colno][1];
                }


 for(i = 0;i<mdeal+1;i++)
        for(colno = 0;colno<ndeal+1;colno++)
                {Ftemp[i*(ndeal+1)+colno][0] = (-1.0/6.0)*Re/deltaT*((18.0)*omegan[i
                    *(ndeal+1)+colno][0] - 9.0*omeganmin1[i*(ndeal+1)+colno][0] +
                    2.0*omeganmin2[i*(ndeal+1)+colno][0]) + Re*(3.0*NLn[i*(ndeal+1)+
```

```
                    colno][0] - 3.0*NLnmin1[i*(ndeal+1)+colno][0] + NLnmin2[i*(ndeal
                    +1)+colno][0]);
                Ftemp[i*(ndeal+1)+colno][1] = (-1.0/6.0)*Re/deltaT*((18.0)*omegan[i*(
                    ndeal+1)+colno][1] - 9.0*omeganmin1[i*(ndeal+1)+colno][1] + 2.0*
                    omeganmin2[i*(ndeal+1)+colno][1]) + Re*(3.0*NLn[i*(ndeal+1)+colno
                    ][1] - 3.0*NLnmin1[i*(ndeal+1)+colno][1] + NLnmin2[i*(ndeal+1)+
                    colno][1]);
                }

copycomplex(NLnmin1, NLnmin2, (mdeal+1)*(ndeal+1));
copycomplex(NLn, NLnmin1, (mdeal+1)*(ndeal+1));

copycomplex(omeganmin1, omeganmin2, (mdeal+1)*(ndeal+1));
copycomplex(omegan, omeganmin1, (mdeal+1)*(ndeal+1));

 chebforcol2Dcpar(Ftemp,temp1,inchebdeal,outchebdeal,pchebfordeal,mdeal,ndeal,
     MPI_complex);
 fourforrow2Dcpar(temp1,temp2,indeal,outdeal,pfordeal,mdeal,ndeal, MPI_complex);

 changeres(temp2, F, mdeal+1, ndeal+1, m+1, n+1);
//===================Setting Boundary Condition = 0.0==================
 for(colno = 0;colno<n+1;colno++)
    {F[(m-1)*(n+1)+colno][0] = F[(m-1)*(n+1)+colno][1] = 0.0;
     F[(m)*(n+1)+colno][0] = F[(m)*(n+1)+colno][1] = 0.0;
     }

#ifdef DEBUG
{ printf("\nF:\n"); print2Dc(F,m+1,n+1);}
#endif

 fftw_free(u);
// fftw_free(v);
 fftw_free(domegadk);
 fftw_free(domegad);
 fftw_free(temp1);
 fftw_free(temp2);
 fftw_free(temp3);
 fftw_free(Ftemp);
 fftw_free(psikndeal);
 fftw_free(omegakndeal);
 free(procarray);
//printf("\n#################\n");
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

/*
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Calculates omegak1 or omegak2 for 1st order BE
int calcBEomegak12(fftw_complex *omegak, double *boundaryval, double Re, double
    deltaT,fftw_complex* in,fftw_complex*out, fftw_plan pback, fftw_plan pfor,
    fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebback, double Lcheb,
    double Lfour, int m, int n, int n1,int n2, int rank,int *tasks,int *tasksnum, int
     numtasks,int numprocs, MPI_Datatype MPI_complex)
{int i,j,l,k,colno;
 fftw_complex *K, *F, *Ftemp, *omegaktemp,*b,*bk, *Keven, *Kodd, *Feven, *Fodd, *
     ueven, *uodd;
 fftw_complex *temp1, *temp2;



 int meven, modd;
 int leven, lodd;
 if((m+1)/2.0 != (int)((m+1)/2.0 ))
```

```
        {meven = ceil1(m+1,2); modd = (m+1)/2;}
 else
        {modd = ceil1(m+1,2); meven = (m+1)/2;}

 Feven = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(meven));
 Fodd = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(modd));

 ueven = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(meven));
 uodd = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(modd));

 Keven = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(meven)*(meven));
 Kodd = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(modd)*(modd));

 temp1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 temp2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
// K = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(m+1));
// F = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 Ftemp =  (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));
 omegaktemp =  (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));
 b = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*2);
 bk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*2);

 makezeroc(b,2*(n+1));
 makezeroc(bk,2*(n+1));

 for(i = 0;i<2*(n+1);i++)
        {bk[i][0] = boundaryval[i];
        bk[i][1] = boundaryval[i];
        }

#ifdef DEBUG
if(rank == 0){ printf("\nboundvals:\n");print2Dc(bk,2,n+1);}
#endif

 for(colno = 0; colno<ceil1(n+1,2)+1; colno++)
    {//asmbKBE(K, Re, deltaT,colno,Lcheb,Lfour, m,n);
     for(i = 2;i<m+1;i++)
         {Ftemp[i][0] = 0.0; Ftemp[i][1] = 0.0;}
     Ftemp[0][0] = bk[colno][0]; Ftemp[0][1] = bk[colno][1];
     Ftemp[1][0] = bk[n+1+colno][0]; Ftemp[1][1] = bk[n+1+colno][1];


        asmbKBEred(Kodd, Keven, Re, deltaT, colno, Lcheb, Lfour, m, n);

//if(rank == 0){printf("\nKodd:\n");print2Dc(Kodd,modd,modd);printf("\nKeven:\n");
    print2Dc(Keven,meven,meven);}

    leven = 1;
    lodd = 1;
        makezeroc(Feven, meven);
        makezeroc(Fodd,modd);
        Fodd[0][0] = Feven[0][0]= (Ftemp[0][0]+Ftemp[1][0])/2.0;
        Fodd[0][1] = Feven[0][1]= (Ftemp[0][1]+Ftemp[1][1])/2.0;


     matrixgausscrim(Keven,ueven,Feven,0,meven);
     matrixgausscrim(Kodd,uodd,Fodd,0,modd);

leven = 0;
lodd = 0;
        for(l = 0;l<m+1;)
                {if(l/2.0 == (int)(l/2.0))
                {if(leven<meven)
                        {omegaktemp[l][0] = ueven[leven][0];
```

```
                        leven++;
                        l++;
                        }
                }
                else
                {
                if(lodd<modd)
                        {omegaktemp[l][0] = uodd[lodd][0];
                        lodd++;
                        l++;
                        }
                }
                }

        if(colno != 0 && colno != ceil1(n+1,2))
             {matrixgausscrim(Keven,ueven,Feven,1,meven);
             matrixgausscrim(Kodd,uodd,Fodd,1,modd);
                leven = 0;
                lodd = 0;
                for(l = 0;l<m+1;)
                        {if(l/2.0 == (int)(l/2.0))
                        {if(leven<meven)
                                {omegaktemp[l][1] = ueven[leven][1];
                                leven++;
                                l++;
                                }
                        }
                        else
                        {
                        if(lodd<modd)
                                {omegaktemp[l][1] = uodd[lodd][1];
                                lodd++;
                                l++;
                                }
                        }

                }
                }

     else
                for(i = 0; i<m+1;i++)
                        omegaktemp[i][1] = 0.0;


//if(rank ==0){printf("\ncolumn no: %d\n",colno);printf("\nomegaodd: ");print2Dc(uodd
    ,1,modd);printf("\nomegaeven: ");print2Dc(ueven,1,meven);printf("\nomegaktemp\n")
    ;print2Dc(omegaktemp,1,m+1);}

/*    matrixgausscrim(K,omegaktemp, Ftemp,0, m+1);
        if(colno != 0 && colno != ceil1(n+1,2))
                matrixgausscrim(K,omegaktemp, Ftemp,1, m+1);
     else
                for(i = 0; i<m+1;i++)
                        omegaktemp[i][1] = 0.0;
*/
/*
     for(i = 0;i<m+1;i++)
         {omegak[i*(n+1)+colno][0] = omegaktemp[i][0]; omegak[i*(n+1)+colno][1] =
             omegaktemp[i][1];
         }
    }

// Do the complex conjugate thing
        for(colno = 1; colno<ceil1(n+1,2);colno++)
```

```
                for(j =0;j<m+1;j++)
                        {omegak[j*(n+1)+(n+1)-colno][0] = omegak[j*(n+1)+colno][0];
                         omegak[j*(n+1)+(n+1)-colno][1] = -omegak[j*(n+1)+colno][1];
                        }

#ifdef DEBUG
if(rank == 0){ printf("Omegak12:\n");print2Dc(omegak,m+1,n+1)
    ;}//////////////////////////
#endif
// chebbackcol2Dcpar(omegak, temp2, incheb, outcheb, pchebback, m, n, MPI_complex);
#ifdef DEBUG
if(rank == 0){ printf("\nOmegak after chebback: i.e. Fourier Coeffs:\n"); print2Dc(
    temp2,m+1,n+1);}////////////////////////////
#endif

 fftw_free(Keven);
 fftw_free(Kodd);
 fftw_free(ueven);
 fftw_free(uodd);
 fftw_free(Feven);
 fftw_free(Fodd);

 fftw_free(temp1);
 fftw_free(temp2);
// fftw_free(K);
// fftw_free(F);
 fftw_free(Ftemp);
 fftw_free(omegaktemp);
 fftw_free(b);
 fftw_free(bk);

 return 1;
}*/
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Calculates omegak1 or omegak2 for 1st order BE
int calcBEomegak12(fftw_complex *omegak, double *boundaryval, double Re, double
    deltaT,fftw_complex* in,fftw_complex*out, fftw_plan pback, fftw_plan pfor,
    fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebback, double Lcheb,
    double Lfour, int m, int n, int n1,int n2, int rank,int *tasks,int *tasksnum, int
     numtasks,int numprocs, MPI_Datatype MPI_complex)
{int i,j,colno;
 fftw_complex *K, *F, *Ftemp, *omegaktemp,*b,*bk;
 fftw_complex *temp1, *temp2;


 temp1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 temp2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 K = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(m+1));
 F = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 Ftemp =  (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));
 omegaktemp =  (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));
 b = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*2);
 bk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*2);

 makezeroc(b,2*(n+1));
 makezeroc(bk,2*(n+1));

 for(i = 0;i<2*(n+1);i++)
        {bk[i][0] = boundaryval[i];
        bk[i][1] = boundaryval[i];
        }
```

```
#ifdef DEBUG
if(rank == 0){ printf("\nboundvals:\n");print2Dc(bk,2,n+1);}
#endif

 for(colno = 0; colno<ceil1(n+1,2)+1; colno++)
    {asmbKBE(K, Re, deltaT,colno,Lcheb,Lfour, m,n);
     for(i = 0;i<m-2+1;i++)
         {Ftemp[i][0] = 0.0; Ftemp[i][1] = 0.0;}
     Ftemp[m-2+1][0] = bk[colno][0]; Ftemp[m-2+1][1] = bk[colno][1];
     Ftemp[m-1+1][0] = bk[n+1+colno][0]; Ftemp[m-1+1][1] = bk[n+1+colno][1];
     matrixgausscrim(K,omegaktemp, Ftemp,0, m+1);
        if(colno != 0 && colno != ceil1(n+1,2))
                matrixgausscrim(K,omegaktemp, Ftemp,1, m+1);
     else
                for(i = 0; i<m+1;i++)
                        omegaktemp[i][1] = 0.0;

     for(i = 0;i<m+1;i++)
         {omegak[i*(n+1)+colno][0] = omegaktemp[i][0]; omegak[i*(n+1)+colno][1] =
             omegaktemp[i][1];
         }
    }

// Do the complex conjugate thing
        for(colno = 1; colno<ceil1(n+1,2);colno++)
                for(j =0;j<m+1;j++)
                        {omegak[j*(n+1)+(n+1)-colno][0] = omegak[j*(n+1)+colno][0];
                         omegak[j*(n+1)+(n+1)-colno][1] = -omegak[j*(n+1)+colno][1];
                        }

#ifdef DEBUG
if(rank == 0){ printf("Omegak12:\n");print2Dc(omegak,m+1,n+1)
    ;}//////////////////////////
#endif
 chebbackcol2Dcpar(omegak, temp2, incheb, outcheb, pchebback, m, n, MPI_complex);
#ifdef DEBUG
if(rank == 0){ printf("\nOmegak after chebback: i.e. Fourier Coeffs:\n"); print2Dc(
    temp2,m+1,n+1);}//////////////////////////
#endif

 fftw_free(temp1);
 fftw_free(temp2);
 fftw_free(K);
 fftw_free(F);
 fftw_free(Ftemp);
 fftw_free(omegaktemp);
 fftw_free(b);
 fftw_free(bk);

 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++



//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Calculates omegak1 or omegak2 for 2nd order BE
int calc2BEomegak12(fftw_complex *omegak, double *boundaryval, double Re, double
    deltaT,fftw_complex* in,fftw_complex*out, fftw_plan pback, fftw_plan pfor,
    fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebback, double Lcheb,
    double Lfour, int m, int n, int n1,int n2, int rank,int *tasks,int *tasksnum, int
     numtasks,int numprocs, MPI_Datatype MPI_complex)
{int i,j,colno;
```

```
 fftw_complex *K, *F, *Ftemp, *omegaktemp,*b,*bk;                                                    1567
 fftw_complex *temp1, *temp2;                                                                        1568
                                                                                                     1569
                                                                                                     1570
 temp1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));                               1571
 temp2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));                               1572
 K = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(m+1));                                   1573
 F = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));                                   1574
 Ftemp =  (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));                                    1575
 omegaktemp =  (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));                               1576
 b = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*2);                                       1577
 bk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*2);                                      1578
                                                                                                     1579
 makezeroc(b,2*(n+1));                                                                               1580
 makezeroc(bk,2*(n+1));                                                                              1581
                                                                                                     1582
 for(i = 0;i<2*(n+1);i++)                                                                            1583
        {bk[i][0] = boundaryval[i];                                                                  1584
        bk[i][1] = boundaryval[i];                                                                   1585
        }                                                                                            1586
                                                                                                     1587
#ifdef DEBUG                                                                                         1588
if(rank == 0){ printf("\nboundvals:\n");print2Dc(bk,2,n+1);}                                         1589
#endif                                                                                               1590
                                                                                                     1591
 for(colno = 0; colno<ceil1(n+1,2)+1; colno++)                                                       1592
    {asmbK2BE(K, Re, deltaT,colno,Lcheb,Lfour, m,n);                                                 1593
     for(i = 0;i<m-2+1;i++)                                                                          1594
         {Ftemp[i][0] = 0.0; Ftemp[i][1] = 0.0;}                                                     1595
     Ftemp[m-2+1][0] = bk[colno][0]; Ftemp[m-2+1][1] = bk[colno][1];                                 1596
     Ftemp[m-1+1][0] = bk[n+1+colno][0]; Ftemp[m-1+1][1] = bk[n+1+colno][1];                         1597
     matrixgausscrim(K,omegaktemp, Ftemp,0, m+1);                                                    1598
        if(colno != 0 && colno != ceil1(n+1,2))                                                      1599
                matrixgausscrim(K,omegaktemp, Ftemp,1, m+1);                                         1600
     else                                                                                            1601
                for(i = 0; i<m+1;i++)                                                                1602
                        omegaktemp[i][1] = 0.0;                                                      1603
                                                                                                     1604
     for(i = 0;i<m+1;i++)                                                                            1605
         {omegak[i*(n+1)+colno][0] = omegaktemp[i][0]; omegak[i*(n+1)+colno][1] =                     1606
             omegaktemp[i][1];
         }                                                                                           1607
    }                                                                                                1608
                                                                                                     1609
// Do the complex conjugate thing                                                                    1610
        for(colno = 1; colno<ceil1(n+1,2);colno++)                                                   1611
                for(j =0;j<m+1;j++)                                                                  1612
                        {omegak[j*(n+1)+(n+1)-colno][0] = omegak[j*(n+1)+colno][0];                   1613
                         omegak[j*(n+1)+(n+1)-colno][1] = -omegak[j*(n+1)+colno][1];                  1614
                        }                                                                            1615
                                                                                                     1616
#ifdef DEBUG                                                                                         1617
if(rank == 0){ printf("Omegak12:\n");print2Dc(omegak,m+1,n+1)                                        1618
    ;}//////////////////////////
#endif                                                                                               1619
 chebbackcol2Dcpar(omegak, temp2, incheb, outcheb, pchebback, m, n, MPI_complex);                    1620
#ifdef DEBUG                                                                                         1621
if(rank == 0){ printf("\nOmegak after chebback: i.e. Fourier Coeffs:\n"); print2Dc(                  1622
    temp2,m+1,n+1);}////////////////////////////
#endif                                                                                               1623
                                                                                                     1624
 fftw_free(temp1);                                                                                   1625
 fftw_free(temp2);                                                                                   1626
 fftw_free(K);                                                                                       1627
```

```
 fftw_free(F);
 fftw_free(Ftemp);
 fftw_free(omegaktemp);
 fftw_free(b);
 fftw_free(bk);

 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Calculates omegak1 or omegak2 for 2nd order BE
int calc3BEomegak12(fftw_complex *omegak, double *boundaryval, double Re, double
    deltaT,fftw_complex* in,fftw_complex*out, fftw_plan pback, fftw_plan pfor,
    fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebback, double Lcheb,
    double Lfour, int m, int n, int n1,int n2, int rank,int *tasks,int *tasksnum, int
     numtasks,int numprocs, MPI_Datatype MPI_complex)
{int i,j,colno;
 fftw_complex *K, *F, *Ftemp, *omegaktemp,*b,*bk;
 fftw_complex *temp1, *temp2;


 temp1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 temp2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 K = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(m+1));
 F = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 Ftemp =  (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));
 omegaktemp =  (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));
 b = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*2);
 bk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*2);

 makezeroc(b,2*(n+1));
 makezeroc(bk,2*(n+1));

 for(i = 0;i<2*(n+1);i++)
        {bk[i][0] = boundaryval[i];
        bk[i][1] = boundaryval[i];
        }

#ifdef DEBUG
if(rank == 0){ printf("\nboundvals:\n");print2Dc(bk,2,n+1);}
#endif

 for(colno = 0; colno<ceil1(n+1,2)+1; colno++)
    {asmbK3BE(K, Re, deltaT,colno,Lcheb,Lfour, m,n);
     for(i = 0;i<m-2+1;i++)
         {Ftemp[i][0] = 0.0; Ftemp[i][1] = 0.0;}
     Ftemp[m-2+1][0] = bk[colno][0]; Ftemp[m-2+1][1] = bk[colno][1];
     Ftemp[m-1+1][0] = bk[n+1+colno][0]; Ftemp[m-1+1][1] = bk[n+1+colno][1];
     matrixgausscrim(K,omegaktemp, Ftemp,0, m+1);
        if(colno != 0 && colno != ceil1(n+1,2))
                matrixgausscrim(K,omegaktemp, Ftemp,1, m+1);
     else
                for(i = 0; i<m+1;i++)
                        omegaktemp[i][1] = 0.0;

     for(i = 0;i<m+1;i++)
         {omegak[i*(n+1)+colno][0] = omegaktemp[i][0]; omegak[i*(n+1)+colno][1] =
             omegaktemp[i][1];
         }
    }

// Do the complex conjugate thing
        for(colno = 1; colno<ceil1(n+1,2);colno++)
```

```
                for(j =0;j<m+1;j++)
                        {omegak[j*(n+1)+(n+1)-colno][0] = omegak[j*(n+1)+colno][0];
                         omegak[j*(n+1)+(n+1)-colno][1] = -omegak[j*(n+1)+colno][1];
                        }

#ifdef DEBUG
if(rank == 0){ printf("Omegak12:\n");print2Dc(omegak,m+1,n+1)
    ;}//////////////////////////
#endif
 chebbackcol2Dcpar(omegak, temp2, incheb, outcheb, pchebback, m, n, MPI_complex);
#ifdef DEBUG
if(rank == 0){ printf("\nOmegak after chebback: i.e. Fourier Coeffs:\n"); print2Dc(
    temp2,m+1,n+1);}////////////////////////////
#endif

 fftw_free(temp1);
 fftw_free(temp2);
 fftw_free(K);
 fftw_free(F);
 fftw_free(Ftemp);
 fftw_free(omegaktemp);
 fftw_free(b);
 fftw_free(bk);

 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Calculates Omegatildek using Second Order Backward Euler:

int calcBEomegaktilde(fftw_complex *omegaktilde, fftw_complex *psik, fftw_complex *
    omegak, fftw_complex *in, fftw_complex *out, fftw_plan pfor, fftw_plan pback,
    fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebfor, fftw_plan
    pchebback, fftw_complex *indeal, fftw_complex *outdeal, fftw_plan pfordeal,
    fftw_plan pbackdeal, fftw_complex *inchebdeal, fftw_complex *outchebdeal,
    fftw_plan pchebfordeal, fftw_plan pchebbackdeal,  double deltaT, double Re,
    double Lcheb,double Lfour, int mdeal, int ndeal, int m, int n, int n1,int n2, int
     rank,int *tasks,int *tasksnum, int numtasks,int numprocs, MPI_Datatype
    MPI_complex)
{int i,j,k,l,colno;
 fftw_complex *Keven, *Kodd, *Ftemp,*omegaktemp,*F,*omegaktildetemp,*omegaktilde2,*
     Feven, *Fodd, *uodd, *ueven;

static MPI_Status Stat;

 double Pk, Qk, Rk;
 int meven, modd;
 int leven, lodd;
 if((m+1)/2.0 != (int)((m+1)/2.0 ))
        {meven = ceil1(m+1,2); modd = (m+1)/2;}
 else
        {modd = ceil1(m+1,2); meven = (m+1)/2;}

 Feven = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(meven));
 Fodd = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(modd));

 ueven = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(meven));
 uodd = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(modd));

 Keven = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(meven)*(meven));
 Kodd = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(modd)*(modd));
 Ftemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));
 omegaktemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));
```

```
 F = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));                                    1740
 omegaktildetemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n2-n1+1));                  1741
 omegaktilde2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(ceil1(n+1,2)                   1742
     +1));
                                                                                                      1743
 assembleFBE(F, psik, omegak, in, out, pfor, pback, incheb, outcheb, pchebfor,                        1744
     pchebback, indeal, outdeal, pfordeal, pbackdeal, inchebdeal, outchebdeal,
     pchebfordeal, pchebbackdeal, deltaT, Re, Lcheb, Lfour, mdeal, ndeal, m, n,
     MPI_complex);
                                                                                                      1745
if(rank<numtasks)                                                                                     1746
{makezeroc(omegaktildetemp,(m+1)*(n2-n1+1));                                                          1747
 k = 0;                                                                                               1748
 for(colno = n1; colno<n2+1;colno++)                                                                  1749
    {//asmbKBE(K, Re, deltaT,colno,Lcheb, Lfour, m, n);                                              1750
        asmbKBEred(Kodd, Keven, Re, deltaT, colno, Lcheb, Lfour, m, n);                               1751
                                                                                                      1752
    leven = 1;                                                                                        1753
    lodd = 1;                                                                                         1754
    for(l = 2;l<m+1;l++)                                                                              1755
         {if(l/2.0 == (int)(l/2.0))                                                                   1756
                        {if (l==2) Pk = 2.0/(4.0*l*(l-1));                                            1757
                        else  Pk = 1.0/(4.0*l*(l-1));                                                 1758
                        if (l<m+1-2) Qk = (-1.0)/(2.0*(l*l-1));                                       1759
                        else Qk = 0.0;                                                                1760
                        if (l<m+1-4) Rk = (1.0)/(4.0*l*(l+1));                                        1761
                        else Rk = 0.0;                                                                1762
                                                                                                      1763
                        if(l<m+1-2)                                                                   1764
                                {Feven[leven][0] = Pk*F[(l-2)*(n+1)+colno][0] + Qk*F[                 1765
                                    l*(n+1)+colno][0] + Rk*F[(l+2)*(n+1)+colno][0];
                                Feven[leven][1] = Pk*F[(l-2)*(n+1)+colno][1] + Qk*F[l                 1766
                                    *(n+1)+colno][1] + Rk*F[(l+2)*(n+1)+colno][1];
                                }                                                                     1767
                        else                                                                          1768
                                {Feven[leven][0] = Pk*F[(l-2)*(n+1)+colno][0] + Qk*F[                 1769
                                    l*(n+1)+colno][0];
                                Feven[leven][1] = Pk*F[(l-2)*(n+1)+colno][1] + Qk*F[l                 1770
                                    *(n+1)+colno][1];
                                }                                                                     1771
                                                                                                      1772
                        leven++;                                                                      1773
                        }                                                                             1774
                else                                                                                  1775
                        {if (l==2) Pk = 2.0/(4.0*l*(l-1));                                            1776
                        else  Pk = 1.0/(4.0*l*(l-1));                                                 1777
                        if (l<m+1-2) Qk = (-1.0)/(2.0*(l*l-1));                                       1778
                        else Qk = 0.0;                                                                1779
                        if (l<m+1-4) Rk = (1.0)/(4.0*l*(l+1));                                        1780
                        else Rk = 0.0;                                                                1781
                                                                                                      1782
                        if(l<m+1-2)                                                                   1783
                                {Fodd[lodd][0] = Pk*F[(l-2)*(n+1)+colno][0] + Qk*F[l                  1784
                                    *(n+1)+colno][0] + Rk*F[(l+2)*(n+1)+colno][0];
                                Fodd[lodd][1] = Pk*F[(l-2)*(n+1)+colno][1] + Qk*F[l*(                 1785
                                    n+1)+colno][1] + Rk*F[(l+2)*(n+1)+colno][1];
                                }                                                                     1786
                        else                                                                          1787
                                {Fodd[lodd][0] = Pk*F[(l-2)*(n+1)+colno][0] + Qk*F[l                  1788
                                    *(n+1)+colno][0];
                                Fodd[lodd][1] = Pk*F[(l-2)*(n+1)+colno][1] + Qk*F[l*(                 1789
                                    n+1)+colno][1];
                                }                                                                     1790
                                                                                                      1791
```

```
                        lodd++;                                                          1792
                        }                                                                1793
                }                                                                        1794
        Fodd[0][0] = Fodd[0][1] = Feven[0][0] = Feven[0][1] = 0.0;                       1795
                                                                                         1796
/*      makezeroc(Ftemp,m+1);                                                            1797
     for(i = 0;i<m+1-2;i++)                                                              1798
         {Ftemp[i][0] = F[i*(n+1)+colno][0]; Ftemp[i][1] = F[i*(n+1)+colno][1];}         1799
*/                                                                                       1800
                                                                                         1801
     matrixgausscrim(Keven,ueven,Feven,0,meven);                                         1802
     matrixgausscrim(Kodd,uodd,Fodd,0,modd);                                             1803
                                                                                         1804
leven = 0;                                                                               1805
lodd = 0;                                                                                1806
        for(l = 0;l<m+1;)                                                                1807
                if(l/2.0 == (int)(l/2.0))                                                1808
                {                                                                        1809
                {if(leven<meven)                                                         1810
                        {omegaktemp[l][0] = ueven[leven][0];                             1811
                        leven++;                                                         1812
                        l++;                                                             1813
                        }                                                                1814
                }                                                                        1815
                }                                                                        1816
                else                                                                     1817
                {                                                                        1818
                if(lodd<modd)                                                            1819
                        {omegaktemp[l][0] = uodd[lodd][0];                               1820
                        lodd++;                                                          1821
                        l++;                                                             1822
                        }                                                                1823
                }                                                                        1824
                                                                                         1825
                                                                                         1826
                                                                                         1827
        if(colno != 0 && colno != ceil1(n+1,2))                                          1828
             {matrixgausscrim(Keven,ueven,Feven,1,meven);                                1829
             matrixgausscrim(Kodd,uodd,Fodd,1,modd);                                     1830
                leven = 0;                                                               1831
                lodd = 0;                                                                1832
                for(l = 0;l<m+1;)                                                        1833
                        if(l/2.0 == (int)(l/2.0))                                        1834
                        {if(leven<meven)                                                 1835
                                {omegaktemp[l][1] = ueven[leven][1];                     1836
                                leven++;                                                 1837
                                l++;                                                     1838
                                }                                                        1839
                        }                                                                1840
                        else                                                             1841
                        {                                                                1842
                        if(lodd<modd)                                                    1843
                                {omegaktemp[l][1] = uodd[lodd][1];                       1844
                                lodd++;                                                  1845
                                l++;                                                     1846
                                }                                                        1847
                        }                                                                1848
                                                                                         1849
                }                                                                        1850
                                                                                         1851
     else                                                                                1852
                for(i = 0; i<m+1;i++)                                                    1853
                        omegaktemp[i][1] = 0.0;                                          1854
                                                                                         1855
```

```
     for(i = 0;i<m+1;i++)
         {omegaktildetemp[k*(m+1)+i][0] = omegaktemp[i][0];omegaktildetemp[k*(m+1)+i
             ][1] = omegaktemp[i][1];}
        k++;
    }

//MPI_Barrier(MPI_COMM_WORLD);
//printf("\n%d: omegaktildetemp:\n",rank);print2Dc(omegaktildetemp,m+1,n2-n1+1);
if(rank<numtasks && rank !=0)
 MPI_Send(omegaktildetemp,(m+1)*(n2-n1+1),MPI_complex,0,rank,MPI_COMM_WORLD);
//MPI_Barrier(MPI_COMM_WORLD);
if(rank == 0)
        {for(i = 1;i<numtasks;i++)
                MPI_Recv(&(omegaktilde2[tasks[i]*(m+1)]), (m+1)*tasksnum[i],
                    MPI_complex, i, i, MPI_COMM_WORLD, &Stat);
        for(i = 0; i <(m+1)*(n2-n1+1);i++)
                {omegaktilde2[i][0] = omegaktildetemp[i][0]; omegaktilde2[i][1] =
                    omegaktildetemp[i][1];}
        }
//MPI_Barrier(MPI_COMM_WORLD);
//if(rank == 0){printf("\nomegaktilde2:\n");print2Dc(omegaktilde2,m+1,ceil1(n+1,2)+1)
    ;}
if(rank == 0)
        for(i = 0;i<m+1;i++)
                for(j = 0;j<ceil1(n+1,2)+1;j++)
                        {omegaktilde[i*(n+1)+j][0] = omegaktilde2[j*(m+1)+i][0];
                        omegaktilde[i*(n+1)+j][1] = omegaktilde2[j*(m+1)+i][1];
                        }
//if(rank == 0){printf("\nomegaktilde:\n");print2Dc(omegaktilde,m+1,n+1);}
}

// Do the complex conjugate thing
        for(colno = 1; colno<ceil1(n+1,2);colno++)
                for(j =0;j<m+1;j++)
                        {omegaktilde[j*(n+1)+(n+1)-colno][0] = omegaktilde[j*(n+1)+
                            colno][0];
                         omegaktilde[j*(n+1)+(n+1)-colno][1] = -omegaktilde[j*(n+1)+
                             colno][1];
                        }
//// MPI_Barrier(MPI_COMM_WORLD);
 MPI_Bcast(omegaktilde,(m+1)*(n+1),MPI_complex,0,MPI_COMM_WORLD);

 //printf("\nOmegaktilde:\n");print2Dc(omegaktilde,m+1,n+1);
 fftw_free(Keven);
 fftw_free(Kodd);
 fftw_free(ueven);
 fftw_free(uodd);
 fftw_free(Feven);
 fftw_free(Fodd);

 fftw_free(Ftemp);
 fftw_free(omegaktemp);
 fftw_free(F);
 fftw_free(omegaktildetemp);
 fftw_free(omegaktilde2);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Calculates Omegatildek using Second Order Backward Euler:
```

```
                                                                                                                    1914
int calc2BEomegaktilde(fftw_complex *omegaktilde, fftw_complex *psikn, fftw_complex *                               1915
    omegakn,fftw_complex *omegan, fftw_complex *omeganmin1,fftw_complex *NLn,
    fftw_complex *NLnmin1, fftw_complex *psiknmin1, fftw_complex *omegaknmin1,
    fftw_complex *in, fftw_complex *out, fftw_plan pfor, fftw_plan pback,
    fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebfor, fftw_plan
    pchebback, fftw_complex *indeal, fftw_complex *outdeal, fftw_plan pfordeal,
    fftw_plan pbackdeal, fftw_complex *inchebdeal, fftw_complex *outchebdeal,
    fftw_plan pchebfordeal, fftw_plan pchebbackdeal, double deltaT, double Re, double
     Lcheb,double Lfour, int mdeal, int ndeal, int m, int n, int n1,int n2, int rank,
    int *tasks,int *tasksnum, int numtasks,int numprocs, MPI_Datatype MPI_complex)
{int i,j,k,l,colno;                                                                                                 1916
 fftw_complex *Keven, *Kodd, *Ftemp,*omegaktemp,*F,*omegaktildetemp,*omegaktilde2,*                                 1917
     Feven, *Fodd, *uodd, *ueven;
                                                                                                                    1918
static MPI_Status Stat;                                                                                             1919
                                                                                                                    1920
 double Pk, Qk, Rk;                                                                                                 1921
 int meven, modd;                                                                                                   1922
 int leven, lodd;                                                                                                   1923
 if((m+1)/2.0 != (int)((m+1)/2.0 ))                                                                                 1924
        {meven = ceil1(m+1,2); modd = (m+1)/2;}                                                                     1925
 else                                                                                                               1926
        {modd = ceil1(m+1,2); meven = (m+1)/2;}                                                                     1927
                                                                                                                    1928
 Feven = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(meven));                                                  1929
 Fodd = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(modd));                                                    1930
                                                                                                                    1931
 ueven = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(meven));                                                  1932
 uodd = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(modd));                                                    1933
                                                                                                                    1934
 Keven = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(meven)*(meven));                                          1935
 Kodd = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(modd)*(modd));                                             1936
 Ftemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));                                                    1937
 omegaktemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));                                               1938
 F = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));                                                  1939
 omegaktildetemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n2-n1+1));                                1940
 omegaktilde2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(ceil1(n+1,2)                                 1941
     +1));
                                                                                                                    1942
if(numprocs<5)                                                                                                      1943
 assembleF2BE2(F, psikn,psiknmin1, omegakn, omegaknmin1, in, out, pfor, pback, incheb                               1944
     , outcheb, pchebfor, pchebback, indeal, outdeal, pfordeal, pbackdeal, inchebdeal
     , outchebdeal, pchebfordeal, pchebbackdeal, deltaT, Re, Lcheb, Lfour, mdeal,
     ndeal, m, n, MPI_complex);
else                                                                                                                1945
 assembleF2BE(F, psikn, omegakn, omegan, omeganmin1, NLn, NLnmin1, in, out, pfor,                                   1946
     pback, incheb, outcheb, pchebfor, pchebback, indeal, outdeal, pfordeal,
     pbackdeal, inchebdeal, outchebdeal, pchebfordeal, pchebbackdeal, deltaT, Re,
     Lcheb, Lfour, mdeal, ndeal, m, n, MPI_complex);
//////////////////////////////////////////////////////////////////////                                            1947
                                                                                                                    1948
                                                                                                                    1949
if(rank<numtasks)                                                                                                   1950
{makezeroc(omegaktildetemp,(m+1)*(n2-n1+1));                                                                        1951
 k = 0;                                                                                                             1952
 for(colno = n1; colno<n2+1;colno++)                                                                                1953
    {//asmbK2BE(K, Re, deltaT,colno,Lcheb, Lfour, m, n);                                                           1954
        asmbK2BEred(Kodd, Keven, Re, deltaT, colno, Lcheb, Lfour, m, n);                                            1955
    leven = 1;                                                                                                      1956
    lodd = 1;                                                                                                       1957
    for(l = 2;l<m+1;l++)                                                                                            1958
         {if(l/2.0 == (int)(l/2.0))                                                                                 1959
                        {if (l==2) Pk = 2.0/(4.0*l*(l-1));                                                          1960
```

```
                        else  Pk = 1.0/(4.0*l*(l-1));                                   1961
                        if (l<m+1-2) Qk = (-1.0)/(2.0*(l*l-1));                         1962
                        else Qk = 0.0;                                                  1963
                        if (l<m+1-4) Rk = (1.0)/(4.0*l*(l+1));                          1964
                        else Rk = 0.0;                                                  1965
                                                                                        1966
                        if(l<m+1-2)                                                     1967
                                {Feven[leven][0] = Pk*F[(l-2)*(n+1)+colno][0] + Qk*F[   1968
                                    l*(n+1)+colno][0] + Rk*F[(l+2)*(n+1)+colno][0];
                                Feven[leven][1] = Pk*F[(l-2)*(n+1)+colno][1] + Qk*F[l   1969
                                    *(n+1)+colno][1] + Rk*F[(l+2)*(n+1)+colno][1];
                                }                                                       1970
                        else                                                            1971
                                {Feven[leven][0] = Pk*F[(l-2)*(n+1)+colno][0] + Qk*F[   1972
                                    l*(n+1)+colno][0];
                                Feven[leven][1] = Pk*F[(l-2)*(n+1)+colno][1] + Qk*F[l   1973
                                    *(n+1)+colno][1];
                                }                                                       1974
                                                                                        1975
                        leven++;                                                        1976
                        }                                                               1977
                else                                                                    1978
                        {if (l==2) Pk = 2.0/(4.0*l*(l-1));                              1979
                        else  Pk = 1.0/(4.0*l*(l-1));                                   1980
                        if (l<m+1-2) Qk = (-1.0)/(2.0*(l*l-1));                         1981
                        else Qk = 0.0;                                                  1982
                        if (l<m+1-4) Rk = (1.0)/(4.0*l*(l+1));                          1983
                        else Rk = 0.0;                                                  1984
                                                                                        1985
                        if(l<m+1-2)                                                     1986
                                {Fodd[lodd][0] = Pk*F[(l-2)*(n+1)+colno][0] + Qk*F[l    1987
                                    *(n+1)+colno][0] + Rk*F[(l+2)*(n+1)+colno][0];
                                Fodd[lodd][1] = Pk*F[(l-2)*(n+1)+colno][1] + Qk*F[l*(   1988
                                    n+1)+colno][1] + Rk*F[(l+2)*(n+1)+colno][1];
                                }                                                       1989
                        else                                                            1990
                                {Fodd[lodd][0] = Pk*F[(l-2)*(n+1)+colno][0] + Qk*F[l    1991
                                    *(n+1)+colno][0];
                                Fodd[lodd][1] = Pk*F[(l-2)*(n+1)+colno][1] + Qk*F[l*(   1992
                                    n+1)+colno][1];
                                }                                                       1993
                                                                                        1994
                        lodd++;                                                         1995
                        }                                                               1996
                }                                                                       1997
        Fodd[0][0] = Fodd[0][1] = Feven[0][0] = Feven[0][1] = 0.0;                      1998
                                                                                        1999
/*      makezeroc(Ftemp,m+1);                                                           2000
     for(i = 0;i<m+1-2;i++)                                                             2001
         {Ftemp[i][0] = F[i*(n+1)+colno][0]; Ftemp[i][1] = F[i*(n+1)+colno][1];}        2002
*/                                                                                      2003
                                                                                        2004
     matrixgausscrim(Keven,ueven,Feven,0,meven);                                        2005
     matrixgausscrim(Kodd,uodd,Fodd,0,modd);                                            2006
                                                                                        2007
leven = 0;                                                                              2008
lodd = 0;                                                                               2009
        for(l = 0;l<m+1;)                                                               2010
                if(l/2.0 == (int)(l/2.0))                                               2011
                {if(leven<meven)                                                        2012
                        {omegaktemp[l][0] = ueven[leven][0];                            2013
                        leven++;                                                        2014
                        l++;                                                            2015
                        }                                                               2016
```

```
                }                                                                                    2017
                else                                                                                 2018
                {                                                                                    2019
                if(lodd<modd)                                                                        2020
                        {omegaktemp[l][0] = uodd[lodd][0];                                           2021
                        lodd++;                                                                      2022
                        l++;                                                                         2023
                        }                                                                            2024
                }                                                                                    2025
                                                                                                     2026
                                                                                                     2027
        if(colno != 0 && colno != ceil1(n+1,2))                                                      2028
             {matrixgausscrim(Keven,ueven,Feven,1,meven);                                            2029
             matrixgausscrim(Kodd,uodd,Fodd,1,modd);                                                 2030
                leven = 0;                                                                           2031
                lodd = 0;                                                                            2032
                for(l = 0;l<m+1;)                                                                    2033
                        {if(leven<meven)                                                             2034
                                {omegaktemp[l][1] = ueven[leven][1];                                 2035
                                leven++;                                                             2036
                                l++;                                                                 2037
                                }                                                                    2038
                        if(lodd<modd)                                                                2039
                                {omegaktemp[l][1] = uodd[lodd][1];                                   2040
                                lodd++;                                                              2041
                                l++;                                                                 2042
                                }                                                                    2043
                        }                                                                            2044
                                                                                                     2045
                }                                                                                    2046
                                                                                                     2047
     else                                                                                            2048
                for(i = 0; i<m+1;i++)                                                                2049
                        omegaktemp[i][1] = 0.0;                                                      2050
                                                                                                     2051
     for(i = 0;i<m+1;i++)                                                                            2052
         {omegaktildetemp[k*(m+1)+i][0] = omegaktemp[i][0];omegaktildetemp[k*(m+1)+i                  2053
             ][1] = omegaktemp[i][1];}
        k++;                                                                                         2054
    }                                                                                                2055
                                                                                                     2056
//MPI_Barrier(MPI_COMM_WORLD);                                                                       2057
//printf("\n%d: omegaktildetemp:\n",rank);print2Dc(omegaktildetemp,m+1,n2-n1+1);                     2058
if(rank<numtasks && rank !=0)                                                                        2059
 MPI_Send(omegaktildetemp,(m+1)*(n2-n1+1),MPI_complex,0,rank,MPI_COMM_WORLD);                        2060
//MPI_Barrier(MPI_COMM_WORLD);                                                                       2061
if(rank == 0)                                                                                        2062
        {for(i = 1;i<numtasks;i++)                                                                   2063
                MPI_Recv(&(omegaktilde2[tasks[i]*(m+1)]), (m+1)*tasksnum[i],                         2064
                    MPI_complex, i, i, MPI_COMM_WORLD, &Stat);
        for(i = 0; i <(m+1)*(n2-n1+1);i++)                                                           2065
                {omegaktilde2[i][0] = omegaktildetemp[i][0]; omegaktilde2[i][1] =                    2066
                    omegaktildetemp[i][1];}
        }                                                                                            2067
//MPI_Barrier(MPI_COMM_WORLD);                                                                       2068
//if(rank == 0){printf("\nomegaktilde2:\n");print2Dc(omegaktilde2,m+1,ceil1(n+1,2)+1)                2069
    ;}
if(rank == 0)                                                                                        2070
        for(i = 0;i<m+1;i++)                                                                         2071
                for(j = 0;j<ceil1(n+1,2)+1;j++)                                                      2072
                        {omegaktilde[i*(n+1)+j][0] = omegaktilde2[j*(m+1)+i][0];                     2073
                        omegaktilde[i*(n+1)+j][1] = omegaktilde2[j*(m+1)+i][1];                      2074
                        }                                                                            2075
//if(rank == 0){printf("\nomegaktilde:\n");print2Dc(omegaktilde,m+1,n+1);}                           2076
```

```
}

// Do the complex conjugate thing
        for(colno = 1; colno<ceil1(n+1,2);colno++)
                for(j =0;j<m+1;j++)
                        {omegaktilde[j*(n+1)+(n+1)-colno][0] = omegaktilde[j*(n+1)+
                            colno][0];
                         omegaktilde[j*(n+1)+(n+1)-colno][1] = -omegaktilde[j*(n+1)+
                             colno][1];
                        }
//// MPI_Barrier(MPI_COMM_WORLD);
 MPI_Bcast(omegaktilde,(m+1)*(n+1),MPI_complex,0,MPI_COMM_WORLD);

 //printf("\nOmegaktilde:\n");print2Dc(omegaktilde,m+1,n+1);
 fftw_free(Keven);
 fftw_free(Kodd);
 fftw_free(ueven);
 fftw_free(uodd);
 fftw_free(Feven);
 fftw_free(Fodd);

 fftw_free(Ftemp);
 fftw_free(omegaktemp);
 fftw_free(F);
 fftw_free(omegaktildetemp);
 fftw_free(omegaktilde2);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Calculates Omegatildek using Second Order Backward Euler:
int calc3BEomegaktilde(fftw_complex *omegaktilde, fftw_complex *psikn, fftw_complex *
    omegakn,fftw_complex *omegan, fftw_complex *omeganmin1, fftw_complex *omeganmin2,
     fftw_complex *NLn, fftw_complex *NLnmin1, fftw_complex *NLnmin2, fftw_complex *
    in, fftw_complex *out, fftw_plan pfor, fftw_plan pback, fftw_complex *incheb,
    fftw_complex *outcheb, fftw_plan pchebfor, fftw_plan pchebback, fftw_complex *
    indeal, fftw_complex *outdeal, fftw_plan pfordeal, fftw_plan pbackdeal,
    fftw_complex *inchebdeal, fftw_complex *outchebdeal, fftw_plan pchebfordeal,
    fftw_plan pchebbackdeal, double deltaT, double Re, double Lcheb,double Lfour, int
     mdeal, int ndeal, int m, int n, int n1,int n2, int rank,int *tasks,int *tasksnum
    , int numtasks,int numprocs, MPI_Datatype MPI_complex)
{int i,j,k,colno;
 fftw_complex *K, *Ftemp,*omegaktemp,*F,*omegaktildetemp,*omegaktilde2;

//static int blockcounts[1];
static MPI_Status Stat;
/*static MPI_Datatype MPI_complex,oldtypes[1];
static MPI_Aint offsets[1];
offsets[0] = 0;
blockcounts[0] = 2;
oldtypes[0] = MPI_DOUBLE;


MPI_Type_struct(1,blockcounts,offsets,oldtypes,&MPI_complex);
MPI_Type_commit (&MPI_complex);
*/

 K = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(m+1));
 Ftemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));
 omegaktemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));
 F = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 omegaktildetemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n2-n1+1));
```

```
 omegaktilde2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(ceil1(n+1,2)
     +1));
//if(numprocs<8)
// assembleF2BE2(F, psikn,psiknmin1, omegakn, omegaknmin1, in, out, pfor, pback,
    incheb, outcheb, pchebfor, pchebback, indeal, outdeal, pfordeal, pbackdeal,
    inchebdeal, outchebdeal, pchebfordeal, pchebbackdeal, deltaT, Re, Lcheb, Lfour,
    mdeal, ndeal, m, n, MPI_complex);
//else
 assembleF3BE(F, psikn, omegakn, omegan, omeganmin1, omeganmin2, NLn, NLnmin1,
     NLnmin2, in, out, pfor,  pback, incheb, outcheb, pchebfor, pchebback, indeal,
     outdeal, pfordeal, pbackdeal, inchebdeal, outchebdeal,  pchebfordeal,
     pchebbackdeal, deltaT, Re, Lcheb, Lfour, mdeal, ndeal, m, n, MPI_complex);
////////////////////////////////////////////////////////////////
if(rank<numtasks)
{makezeroc(omegaktildetemp,(m+1)*(n2-n1+1));
 k = 0;
 for(colno = n1; colno<n2+1;colno++)
    {asmbK3BE(K, Re, deltaT,colno,Lcheb, Lfour, m, n);

        makezeroc(Ftemp,m+1);
     for(i = 0;i<m+1-2;i++)
         {Ftemp[i][0] = F[i*(n+1)+colno][0]; Ftemp[i][1] = F[i*(n+1)+colno][1];}

     matrixgausscrim(K,omegaktemp,Ftemp,0,m+1);

        if(colno != 0 && colno != ceil1(n+1,2))
             matrixgausscrim(K,omegaktemp,Ftemp,1,m+1);
     else
                for(i = 0; i<m+1;i++)
                        omegaktemp[i][1] = 0.0;

     for(i = 0;i<m+1;i++)
         {omegaktildetemp[k*(m+1)+i][0] = omegaktemp[i][0];omegaktildetemp[k*(m+1)+i
             ][1] = omegaktemp[i][1];}
        k++;
    }

//MPI_Barrier(MPI_COMM_WORLD);
//printf("\n%d: omegaktildetemp:\n",rank);print2Dc(omegaktildetemp,m+1,n2-n1+1);
if(rank<numtasks && rank !=0)
 MPI_Send(omegaktildetemp,(m+1)*(n2-n1+1),MPI_complex,0,rank,MPI_COMM_WORLD);
//MPI_Barrier(MPI_COMM_WORLD);
if(rank == 0)
        {for(i = 1;i<numtasks;i++)
                MPI_Recv(&(omegaktilde2[tasks[i]*(m+1)]), (m+1)*tasksnum[i],
                    MPI_complex, i, i, MPI_COMM_WORLD, &Stat);
        for(i = 0; i <(m+1)*(n2-n1+1);i++)
                {omegaktilde2[i][0] = omegaktildetemp[i][0]; omegaktilde2[i][1] =
                    omegaktildetemp[i][1];}
        }
//MPI_Barrier(MPI_COMM_WORLD);
//if(rank == 0){printf("\nomegaktilde2:\n");print2Dc(omegaktilde2,m+1,ceil1(n+1,2)+1)
    ;}
if(rank == 0)
        for(i = 0;i<m+1;i++)
                for(j = 0;j<ceil1(n+1,2)+1;j++)
                        {omegaktilde[i*(n+1)+j][0] = omegaktilde2[j*(m+1)+i][0];
                        omegaktilde[i*(n+1)+j][1] = omegaktilde2[j*(m+1)+i][1];
                        }
//if(rank == 0){printf("\nomegaktilde:\n");print2Dc(omegaktilde,m+1,n+1);}
}
////////////////////////////////////////////////////////////////
```

```
/* for(colno = 0; colno<ceil1(n+1,2)+1; colno++)
 //for(colno = 0;colno<n+1;colno++)
    {asmbK2BE(K, Re, deltaT,colno,Lcheb, Lfour, m, n);

        makezeroc(Ftemp,m+1);
     for(i = 0;i<m+1-2;i++)
         {Ftemp[i][0] = F[i*(n+1)+colno][0]; Ftemp[i][1] = F[i*(n+1)+colno][1];}

     matrixgausscrim(K,omegaktemp,Ftemp,0,m+1);

        if(colno != 0 && colno != ceil1(n+1,2))
             matrixgausscrim(K,omegaktemp,Ftemp,1,m+1);
     else
                for(i = 0; i<m+1;i++)
                        omegaktemp[i][1] = 0.0;

     for(i = 0;i<m+1;i++)
         {omegaktilde[i*(n+1)+colno][0] = omegaktemp[i][0];omegaktilde[i*(n+1)+colno
             ][1] = omegaktemp[i][1];}
    }

*/

// Do the complex conjugate thing
        for(colno = 1; colno<ceil1(n+1,2);colno++)
                for(j =0;j<m+1;j++)
                        {omegaktilde[j*(n+1)+(n+1)-colno][0] = omegaktilde[j*(n+1)+
                            colno][0];
                         omegaktilde[j*(n+1)+(n+1)-colno][1] = -omegaktilde[j*(n+1)+
                             colno][1];
                        }
//// MPI_Barrier(MPI_COMM_WORLD);
 MPI_Bcast(omegaktilde,(m+1)*(n+1),MPI_complex,0,MPI_COMM_WORLD);

 //printf("\nOmegaktilde:\n");print2Dc(omegaktilde,m+1,n+1);
 fftw_free(K);
 fftw_free(Ftemp);
 fftw_free(omegaktemp);
 fftw_free(F);
 fftw_free(omegaktildetemp);
 fftw_free(omegaktilde2);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//This File Reads in all the data
int readinput(char *inputfile, int *xres, int *yres,double *xlen,double *ylen,int *xn
    ,int *yn, double *U,double **boundaryval)
{int i;char line[STRMAX];
 FILE *inputfp;
 inputfp = fopen(inputfile, "r");
 if(inputfp ==NULL) return 0;
 fnamesearch("SIZE",":", inputfp);
 (*xres) = fgetdim(inputfp);//getterm(line,STRMAX);(*yres) =atoi(line);getterm(line,
     STRMAX);(*xres) =atoi(line);
 (*yres) = fgetdim(inputfp);
 printf("\nxres: %d\t\tyres: %d\n",(*xres),(*yres));
 (*boundaryval) = malloc(sizeof(double)*(*xres)*2);

 fnamesearch("BOUNDARYVEL",":", inputfp);
        fgetmatrix((*boundaryval),2, *xres, inputfp);
//      printf("\nBoundary Velocities: \n");print2D((*boundaryval),2, *xres);printf
```

```
    ("\n");////////////////
                                                                                2242
 fnamesearch("LENGTHS",":", inputfp);                                           2243
 fgetval(xlen,inputfp);//getterm(line,STRMAX);(*yres) =atoi(line);getterm(line,STRMAX  2244
     );(*xres) =atoi(line);
 fgetval(ylen,inputfp);                                                         2245
 printf("\nxlen: %16.16g\t\tylen: %16.16g\n",(*xlen),(*ylen));                  2246
 (*xn) = fgetdim(inputfp);                                                      2247
 (*yn) = fgetdim(inputfp);                                                      2248
 printf("\nxn = %d \t\tyn = %d\n",(*xn),(*yn));                                 2249
                                                                                2250
 fnamesearch("UMAX",":", inputfp);                                              2251
 fgetval(U,inputfp);                                                            2252
 printf("\nUmax = %16.16g\n",(*U));                                             2253
 fclose(inputfp);                                                               2254
 return 1;                                                                      2255
}                                                                               2256
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  2257
                                                                                2258
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  2259
//Assembles the K matrix for Psi, for cheb columnwise and four rowwise, Note done  2260
    only one column at a time
// Temp Note = Real part = Imag part                                            2261
int asmbKpsikc(fftw_complex *K, int colno,double Lcheb, double Lfour,int m, int n)  2262
{int l,k,p,multiplier,i;                                                        2263
 double ck;                                                                     2264
                                                                                2265
 makezeroc(K, (m+1)*(m+1));                                                     2266
 for(l = 0;l<m-2+1;l++)                                                         2267
         for(p = l+2;p<m+1;p++)                                                 2268
             {if (l==0)ck=2.0; else ck = 1.0;                                   2269
                 if((p+l)/2 == (p+l)/2.0)                                       2270
                     {K[(l+2)*(m+1)+p][0] +=(2.0/Lcheb)*(2.0/Lcheb)*p*(p*p-l*l)/ck; K  2271
                         [(l+2)*(m+1)+p][1] +=(2.0/Lcheb)*(2.0/Lcheb)*p*(p*p-l*l)/ck
                         ;//K[(l+2)*(n+1)+p][1] +=p*(p*p-l*l)/ck;
                     }                                                          2272
             }                                                                  2273
 for(l = 0; l<m-2+1;l++)                                                        2274
      {if(colno<ceil1(n+1,2)) multiplier = colno;else multiplier = (colno-n-1);K[(l  2275
          +2)*(m+1)+l][0]+= (2.0*PI/Lfour)*(2.0*PI/Lfour)*(-(multiplier*1.0)*(
          multiplier*1.0));K[(l+2)*(m+1)+l][1]+= (2.0*PI/Lfour)*(2.0*PI/Lfour)*(-(
          multiplier*1.0)*(multiplier*1.0));}//K[(l+2)*(n+1)+l][1]+= -(colno*1.0)*(
          colno*1.0);}
                                                                                2276
 for(p = 0;p<m+1;p++)                                                           2277
      {K[p][0]=pow(-1,p)*1.0; K[p][1]=pow(-1,p)*1.0;//K[p][1]+=pow(-1,p)*1.0;   2278
       K[(m+1)+p][0]=1.0; K[(m+1)+p][1]=1.0;//K[(n+1)+p][1]+=1.0;               2279
      }                                                                         2280
                                                                                2281
 return 1;                                                                      2282
}                                                                               2283
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  2284
                                                                                2285
                                                                                2286
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  2287
// Calculates psik from omegak                                                  2288
int calcpsik(fftw_complex *psik, fftw_complex *omegak, double Lcheb, double Lfour,  2289
    int m, int n, int n1,int n2, int rank,int *tasks,int *tasksnum, int numtasks,int
    numprocs, MPI_Datatype MPI_complex)
{int i,j,colno,k,rc;                                                            2290
 fftw_complex *K, *F,*psiktemp,*psiktempsend,*psik2,*omegak2;                   2291
                                                                                2292
static MPI_Status Stat;                                                         2293
                                                                                2294
```

```
 K = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(m+1));                                    2295
 F = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));                                          2296
 psiktemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));                                   2297
 psiktempsend = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n2-n1+1));                     2298
 psik2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(ceil1(n+1,2)+1));                     2299
 omegak2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*(m+1));                              2300
                                                                                                      2301
                                                                                                      2302
////////////////////////////////////////////////                                                      2303
                                                                                                      2304
 //MPI_Barrier(MPI_COMM_WORLD);                                                                       2305
// MPI_Bcast(omegak,(m+1)*(n+1),MPI_complex,0,MPI_COMM_WORLD);                                         2306
if(rank<numtasks)                                                                                     2307
{ makezeroc(psiktempsend,(m+1)*(n2-n1+1));                                                            2308
 k = 0;                                                                                               2309
 for(colno = n1; colno<n2+1;colno++)                                                                  2310
    {asmbKpsikc(K,colno,Lcheb, Lfour, m,n);                                                           2311
    for(i = 0;i<m-2+1;i++)                                                                            2312
          {F[i+2][0] = -omegak[i*(n+1)+colno][0]; F[i+2][1] = -omegak[i*(n+1)+colno                   2313
              ][1];}
         F[0][0] = F[0][1] = 0.0;                                                                     2314
      F[1][0] = 0.0;F[1][1] = 0.0;                                                                    2315
     matrixgausscrim(K,psiktemp,F,0,m+1);                                                             2316
                                                                                                      2317
        if(colno != 0 && colno != ceil1(n+1,2))                                                       2318
             matrixgausscrim(K,psiktemp,F,1,m+1);                                                     2319
     else                                                                                             2320
                for(i = 0; i<m+1;i++)                                                                 2321
                        psiktemp[i][1] = 0.0;                                                         2322
                                                                                                      2323
     for(i = 0;i<m+1;i++)                                                                             2324
         {psiktempsend[k*(m+1)+i][0] = psiktemp[i][0];psiktempsend[k*(m+1)+i][1] =                     2325
             psiktemp[i][1];}
        k++;                                                                                          2326
    }                                                                                                 2327
                                                                                                      2328
// MPI_Barrier(MPI_COMM_WORLD);//printf("\n%d: psiktempsend:\n",rank);print2Dc(                        2329
    psiktempsend,m+1,n2-n1+1);
if(rank<numtasks && rank !=0)                                                                         2330
 MPI_Send(psiktempsend,(m+1)*(n2-n1+1),MPI_complex,0,rank,MPI_COMM_WORLD);                            2331
// MPI_Barrier(MPI_COMM_WORLD);                                                                       2332
if(rank == 0)                                                                                         2333
        {for(i = 1;i<numtasks;i++)                                                                    2334
                MPI_Recv(&(psik2[tasks[i]*(m+1)]), (m+1)*tasksnum[i], MPI_complex, i,                 2335
                     i, MPI_COMM_WORLD, &Stat);
        for(i = 0; i <(m+1)*(n2-n1+1);i++)                                                            2336
                {psik2[i][0] = psiktempsend[i][0]; psik2[i][1] = psiktempsend[i][1];}                 2337
        }                                                                                             2338
                                                                                                      2339
//MPI_Barrier(MPI_COMM_WORLD);//if(rank == 0){printf("\npsik2:\n");print2Dc(psik2,m                    2340
    +1,ceil1(n+1,2)+1);}
if(rank == 0)                                                                                         2341
        for(i = 0;i<m+1;i++)                                                                          2342
                for(j = 0;j<ceil1(n+1,2)+1;j++)                                                       2343
                        {psik[i*(n+1)+j][0] = psik2[j*(m+1)+i][0];                                    2344
                        psik[i*(n+1)+j][1] = psik2[j*(m+1)+i][1];                                     2345
                        }//if(rank == 0){printf("\npsik:\n");print2Dc(psik,m+1,n+1);}                 2346
}                                                                                                     2347
                                                                                                      2348
// MPI_Barrier(MPI_COMM_WORLD);                                                                       2349
 MPI_Bcast(psik,(m+1)*(n+1),MPI_complex,0,MPI_COMM_WORLD);                                            2350
                                                                                                      2351
 fftw_free(K);                                                                                        2352
 fftw_free(F);                                                                                        2353
```

```
 fftw_free(psiktemp);                                                                    2354
 fftw_free(psiktempsend);                                                                2355
 fftw_free(psik2);                                                                       2356
 fftw_free(omegak2);                                                                     2357
 return 1;                                                                               2358
}                                                                                        2359
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++        2360
                                                                                         2361
                                                                                         2362
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++        2363
//Calculates psik12                                                                      2364
int calcpsik12(fftw_complex *psik, fftw_complex *omegak, double Lcheb, double Lfour,     2365
    int m, int n, int n1,int n2, int rank,int *tasks,int *tasksnum, int numtasks,int
    numprocs)
{int i,j,colno;                                                                          2366
 fftw_complex *K, *F,*psiktemp;                                                          2367
                                                                                         2368
 K = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(m+1));                       2369
 F = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));                             2370
 psiktemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1));                      2371
                                                                                         2372
 for(colno = 0;colno<n+1;colno++)                                                        2373
     {asmbKpsikc(K,colno,Lcheb, Lfour, m,n);                                             2374
      for(i = 0;i<m-2+1;i++)                                                             2375
          {F[i+2][0] = -omegak[i*(n+1)+colno][0]; F[i+2][1] = -omegak[i*(n+1)+colno     2376
              ][1];}
      F[0][0] = F[0][1] = 0.0;                                                           2377
      F[1][0] = 0.0;F[1][1] = 0.0;                                                       2378
      matrixgausscrim(K, psiktemp, F,0,m+1);                                             2379
         matrixgausscrim(K, psiktemp, F,1,m+1);                                          2380
                                                                                         2381
      for(i = 0; i<m+1; i++)                                                             2382
          {psik[i*(n+1)+colno][0] = psiktemp[i][0]; psik[i*(n+1)+colno][1] = psiktemp    2383
              [i][1];}
      }                                                                                  2384
                                                                                         2385
 fftw_free(K);                                                                           2386
 fftw_free(F);                                                                           2387
 fftw_free(psiktemp);                                                                    2388
 return 1;                                                                               2389
}                                                                                        2390
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++        2391
                                                                                         2392
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++        2393
//Initialises omegak                                                                     2394
 int initomegak(fftw_complex *omegak, fftw_complex *in, fftw_complex *out, fftw_plan     2395
     pfor,fftw_plan pback,fftw_complex *incheb, fftw_complex *outcheb, fftw_plan
     pchebfor, fftw_plan pchebback, double *boundaryval,double Lcheb, int m, int n,
     int n1,int n2, int rank,int *tasks,int *tasksnum, int numtasks,int numprocs,
     MPI_Datatype MPI_complex)
{int i,j,colno,rowno;double L;                                                           2396
 fftw_complex *u, *uk, *ukint;                                                           2397
                                                                                         2398
 u = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));                       2399
 uk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));                      2400
 ukint = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));                   2401
                                                                                         2402
 makezeroc(u, (m+1)*(n+1));                                                              2403
                                                                                         2404
 for(colno = 0;colno<n+1; colno++)                                                       2405
     {u[colno][0] = boundaryval[colno];u[m*(n+1)+colno][0] = boundaryval[(n+1)+colno     2406
         ];}
                                                                                         2407
// for(colno = 0;colno<n+1; colno++)for(rowno = 0;rowno<m+1; rowno++){u[rowno*(n+1)+     2408
```

```
    colno][0] = -cos(1.0*PI*rowno/m)/2.0+0.5;}
 //for(colno = 0;colno<n+1; colno++)for(rowno = 0;rowno<m+1; rowno++){u[rowno*(n+1)+
     colno][0] = -cos(1.0*PI*rowno/(n*1.0))/2.0+0.5;}
// printf("\nINITOMEGAK\n");
#ifdef DEBUG
if(rank == 0){ printf("\nInitial u:\n"); print2Dc(u, m+1,n+1)
    ;}/////////////////////////////////////////////
#endif
 fourforrow2Dcpar (u, ukint, in, out, pfor, m, n, MPI_complex);
 chebforcol2Dcpar (ukint, uk, incheb, outcheb, pchebfor, m, n, MPI_complex);

 getchebu1kcol2Dc (uk, ukint,Lcheb, m, n);

 chebbackcol2Dcpar (ukint, uk, incheb, outcheb, pchebback, m, n, MPI_complex);
 fourbackrow2Dcpar (uk, u, in, out, pback, m, n, MPI_complex);


 for(i = 0;i<(m+1)*(n+1);i++){u[i][0] = -u[i][0]; u[i][1] = -u[i][1];}

 fourforrow2Dcpar (u, ukint, in, out, pfor, m, n, MPI_complex);
 chebforcol2Dcpar (ukint, omegak, incheb, outcheb, pchebfor, m, n, MPI_complex);

 fftw_free(u);
 fftw_free(uk);
 fftw_free(ukint);

 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Calculates the inverse of the influence matrix
// Temp note: This the new one WITH the complex conj stuff
int asmbinfmat(fftw_complex *Ainv, fftw_complex *psik1, fftw_complex *psik2,
    fftw_complex *in, fftw_complex *out, fftw_plan pback,fftw_complex *incheb,
    fftw_complex *outcheb, fftw_plan pchebback, double *boundaryval,double Lcheb,int
    m, int n, int n1,int n2, int rank,int *tasks,int *tasksnum, int numtasks,int
    numprocs, MPI_Datatype MPI_complex)
{int i, j,k, colno;
 fftw_complex *u1, *u2,*temp2D1;
 double detAr, detAi;
 u1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 u2 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 temp2D1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));

 getchebu1kcol2Dc (psik1, temp2D1, Lcheb,m, n);
 chebbackcol2Dcpar(temp2D1,u1,incheb,outcheb,pchebback,m,n, MPI_complex);
#ifdef DEBUG
if(rank == 0){ printf("\nU1 for Ainv:\n");print2Dc(u1, m+1,n+1)
    ;}/////////////////////////////////////
#endif
 getchebu1kcol2Dc (psik2, temp2D1, Lcheb,m, n);//CHECKED
 chebbackcol2Dcpar(temp2D1,u2,incheb,outcheb,pchebback,m,n, MPI_complex);
#ifdef DEBUG
if(rank == 0){ printf("\nU2 for Ainv:\n");print2Dc(u2, m+1,n+1)
    ;}/////////////////////////////////////
#endif
 makezeroc(Ainv,4*(n+1));
 //This calculates the Ainv

 for(colno=0;colno<ceil1(n+1,2)+1;colno++)
     {detAr = u1[colno][0]*u2[m*(n+1)+colno][0] - u1[m*(n+1)+colno][0]*u2[colno][0];
        if(colno !=0 && colno != ceil1(n+1,2))
                detAi = u1[colno][1]*u2[m*(n+1)+colno][1] - u1[m*(n+1)+colno][1]*u2[
```

```
                    colno][1];
#ifdef DEBUG                                                                    2464
if(rank == 0){      printf("\ndetAr: %g \tdetAi: %g\n", detAr,detAi);}          2465
#endif                                                                          2466
      Ainv[colno*4+0][0] = 1.0/detAr*u2[m*(n+1)+colno][0]; Ainv[colno*4+0][1] = 1.0/   2467
          detAi*u2[m*(n+1)+colno][1];
      Ainv[colno*4+1][0] = -1.0/detAr*u2[colno][0]; Ainv[colno*4+1][1] = -1.0/detAi*   2468
          u2[colno][1];
      Ainv[colno*4+2][0] = -1.0/detAr*u1[m*(n+1)+colno][0]; Ainv[colno*4+2][1] =       2469
          -1.0/detAi*u1[m*(n+1)+colno][1];
      Ainv[colno*4+3][0] = 1.0/detAr*u1[colno][0]; Ainv[colno*4+3][1] = 1.0/detAi*u1[  2470
          colno][1];
                                                                                2471
        if(colno !=0 && colno != ceil1(n+1,2))                                  2472
                {Ainv[colno*4+0][1] = 1.0/detAi*u2[m*(n+1)+colno][1];           2473
                Ainv[colno*4+1][1] = -1.0/detAi*u2[colno][1];                   2474
                Ainv[colno*4+2][1] = -1.0/detAi*u1[m*(n+1)+colno][1];           2475
                Ainv[colno*4+3][1] = 1.0/detAi*u1[colno][1];                    2476
                }                                                               2477
        else Ainv[colno*4+0][1] = Ainv[colno*4+1][1] = Ainv[colno*4+2][1] = Ainv[       2478
            colno*4+3][1] = 0.0;
                                                                                2479
     }                                                                          2480
                                                                                2481
for(colno = 1;  colno<ceil1(n+1,2);colno++)                                     2482
                {Ainv[(n+1 -colno)*4+0][0] = Ainv[colno*4+0][0];                2483
                Ainv[(n+1 -colno)*4+1][0] = Ainv[colno*4+1][0];                 2484
                Ainv[(n+1 -colno)*4+2][0] = Ainv[colno*4+2][0];                 2485
                Ainv[(n+1 -colno)*4+3][0] = Ainv[colno*4+3][0];                 2486
                                                                                2487
                Ainv[(n+1 -colno)*4+0][1] = -Ainv[colno*4+0][1];                2488
                Ainv[(n+1 -colno)*4+1][1] = -Ainv[colno*4+1][1];                2489
                Ainv[(n+1 -colno)*4+2][1] = -Ainv[colno*4+2][1];                2490
                Ainv[(n+1 -colno)*4+3][1] = -Ainv[colno*4+3][1];                2491
                }                                                               2492
                                                                                2493
 fftw_free(u1);                                                                 2494
 fftw_free(u2);                                                                 2495
 fftw_free(temp2D1);                                                            2496
                                                                                2497
 return 1;                                                                      2498
}                                                                               2499
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++   2500
                                                                                2501
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++   2502
//Calculates Lambda from the Influence Matrix                                   2503
int calclambda(fftw_complex *lambda, fftw_complex *psiktilde, fftw_complex *Ainv,   2504
    fftw_complex *in, fftw_complex *out, fftw_plan pfor, fftw_plan pback,fftw_complex
     *incheb, fftw_complex *outcheb, fftw_plan pchebback, double *boundval,double
    Lcheb, int m, int n, int n1,int n2, int rank,int *tasks,int *tasksnum, int
    numtasks,int numprocs, MPI_Datatype MPI_complex)
{int i, j, k,colno;                                                             2505
 fftw_complex *u, *lambdatemp, *utemp,*Atemp,*b,*bk,*temp2D1;                   2506
                                                                                2507
 u = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));              2508
 lambdatemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*2);               2509
                                                                                2510
 utemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*2);                    2511
 Atemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*4);                    2512
 b = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*2);                  2513
 bk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*2);                 2514
  temp2D1 = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));       2515
                                                                                2516
#ifdef DEBUG                                                                   2517
```

```
if(rank == 0){ printf("\nCALCLAMBDA:\n"); }
#endif
 makezeroc(b,2*(n+1));
 makezeroc(lambdatemp,2);

 for(i = 0;i<2*(n+1);i++)
        b[i][0] = boundval[i];

 fourforrow2Dcpar(b,bk,in,out,pfor,1,n, MPI_complex);

#ifdef DEBUG
if(rank == 0){ printf("\nboundval:\n");print2Dc(bk,2,n+1);}
#endif

 getchebu1kcol2Dc (psiktilde, temp2D1, Lcheb,m, n);//CHECKED
 chebbackcol2Dcpar(temp2D1,u,incheb,outcheb,pchebback,m,n, MPI_complex);

#ifdef DEBUG
if(rank == 0){ printf("\npsiprime from psiktilde (i.e. fourier coeffs):\n");print2Dc(
    u,m+1,n+1);}////////////////////////////////////
#endif

 for(colno = 0; colno<n+1; colno++)
     {utemp[0][0] = bk[colno][0] - u[colno][0];utemp[0][1] =bk[colno][1] - u[colno
         ][1];
      utemp[1][0] = bk[(n+1)+colno][0]- u[m*(n+1)+colno][0];utemp[1][1] =bk[(n+1)+
          colno][1]- u[m*(n+1)+colno][1];

            for(i = 0;i<4;i++){Atemp[i][0]=Ainv[colno*4+i][0];Atemp[i][1]=Ainv[colno
                *4+i][1];}
      matrixmultc1 (Atemp, utemp,lambdatemp,2,2,1);

#ifdef DEBUG
if(rank == 0){ printf("utemp %d: ",colno); print2Dc(utemp,1,2); printf("Ainv:\t");
    print2Dc(&(Ainv[colno*4]),1,4);   printf("lambdatemp %d: ",colno); print2Dc(
    lambdatemp,1,2); }  /////////
#endif

      lambda[colno][0] = lambdatemp[0][0];lambda[colno][1] = lambdatemp[0][1];
      lambda[(n+1)+colno][0] = lambdatemp[1][0];lambda[(n+1)+colno][1] = lambdatemp
          [1][1];

     }

 fftw_free(u);
 fftw_free(lambdatemp);
 fftw_free(utemp);
 fftw_free(Atemp);
 fftw_free(b);
 fftw_free(bk);
 fftw_free(temp2D1);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Calculates the coeffs for omegak from omegaktilde, omegak1, omegak2. Can be used to
     determine psik as well
int calcomegak(fftw_complex *omegak, fftw_complex *omegaktilde, fftw_complex *omegak1
    , fftw_complex *omegak2, fftw_complex *lambda, fftw_complex *incheb, fftw_complex
     *outcheb, fftw_plan pchebfor, fftw_plan pchebback, int m, int n, int n1,int n2,
    int rank,int *tasks,int *tasksnum, int numtasks,int numprocs, MPI_Datatype
    MPI_complex)
{int i,j,k,colno;
```

```
 fftw_complex *omega1,*omega2,*omegatilde,*omega;                                    2570
                                                                                    2571
 omega  = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));             2572
 omega1  = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));            2573
 omega2  = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));            2574
 omegatilde  = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));        2575
                                                                                    2576
                                                                                    2577
 chebbackcol2Dcpar(omegaktilde, omegatilde, incheb, outcheb, pchebback, m,n,        2578
     MPI_complex);
 chebbackcol2Dcpar(omegak1, omega1, incheb, outcheb, pchebback, m,n, MPI_complex);  2579
 chebbackcol2Dcpar(omegak2, omega2, incheb, outcheb, pchebback, m,n, MPI_complex);  2580
                                                                                    2581
 for(colno = 0;colno<n+1;colno++)                                                   2582
     {for(i=0;i<m+1;i++)                                                            2583
          {omega[i*(n+1)+colno][0] = omegatilde[i*(n+1)+colno][0]+ lambda[colno][0]* 2584
              omega1[i*(n+1)+colno][0] + lambda[(n+1)+colno][0]*omega2[i*(n+1)+colno
              ][0];
                omega[i*(n+1)+colno][1] = omegatilde[i*(n+1)+colno][1]+ lambda[colno 2585
                    ][1]*omega1[i*(n+1)+colno][1] + lambda[(n+1)+colno][1]*omega2[i*(
                    n+1)+colno][1];
          }                                                                         2586
     }                                                                              2587
 chebforcol2Dcpar(omega, omegak, incheb, outcheb, pchebfor, m, n, MPI_complex);     2588
                                                                                    2589
 fftw_free(omega);                                                                  2590
 fftw_free(omega1);                                                                 2591
 fftw_free(omega2);                                                                 2592
 fftw_free(omegatilde);                                                             2593
 return 1;                                                                          2594
}                                                                                   2595
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++   2596
                                                                                    2597
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++   2598
// Calculates u and v for RK4                                                       2599
int calcuvRK4(fftw_complex* psik, double **uRK4, double **uRK4i, double **vRK4,     2600
    double **vRK4i, double Lcheb, double Lfour, int RK4size, int m, int n)
{int i,j;                                                                           2601
 double *temp;                                                                      2602
 fftw_complex *uk;                                                                  2603
                                                                                    2604
 uk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));                 2605
 temp = malloc(sizeof(double)*(m+1)*(n+1));                                         2606
                                                                                    2607
                                                                                    2608
 for(i = RK4size-1; i>0;i--)                                                        2609
        {copyarray(uRK4[i-1],uRK4[i],(m+1)*(n+1));}                                 2610
 for(i = RK4size-1; i>0;i--)                                                        2611
        {copyarray(uRK4i[i-1],uRK4i[i],(m+1)*(n+1));}                               2612
                                                                                    2613
                                                                                    2614
 for(i = RK4size-1; i>0;i--)                                                        2615
        {copyarray(vRK4[i-1],vRK4[i],(m+1)*(n+1));}                                 2616
 for(i = RK4size-1; i>0;i--)                                                        2617
        {copyarray(vRK4i[i-1],vRK4i[i],(m+1)*(n+1));}                               2618
                                                                                    2619
 getchebu1kcol2Dc (psik, uk,Lcheb, m, n);                                           2620
 for(i = 0;i<(m+1)*(n+1);i++)                                                       2621
        {uRK4[0][i] = uk[i][0];                                                     2622
        uRK4i[0][i] = uk[i][1];                                                     2623
        }                                                                           2624
//printf("\nuk:\n");print2Dc(uk,(m+1),(n+1));                                       2625
 getfouru1krow2Dc (psik, uk,Lfour, m, n);                                           2626
 for(i = 0;i<(m+1)*(n+1);i++)                                                       2627
```

```
        {vRK4[0][i] = (-1.0) * uk[i][0];                                                    2628
        vRK4i[0][i] = (-1.0) * uk[i][1];                                                    2629
        }                                                                                   2630
                                                                                            2631
 fftw_free(uk);                                                                             2632
 free(temp);                                                                                2633
 return 1;                                                                                  2634
}                                                                                           2635
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++   2636
                                                                                            2637
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++   2638
int printpsiuv1(fftw_complex* psik, double *outputcount, char* outstr, char *option,       2639
    char *valoutput, int coordx, int coordy, long unsigned int coordn, double *coord,
    double Lcheb, double Lfour, int m, int n, int rank, int numprocs, int* eventflag,
     double nondimtime, double dimtime, MPI_Datatype MPI_complex)
{char outstr1[STRMAX*5], outstr2[STRMAX*5];                                                 2640
 FILE *outputTfp,*outputTfp2;                                                               2641
 fftw_complex *uk, *vk;                                                                     2642
 double *psi, *u, *v,*psi2,*u2,*v2;                                                         2643
 int i,j;                                                                                   2644
                                                                                            2645
                                                                                            2646
 uk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));                         2647
 vk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));                         2648
 psi = malloc(sizeof(double)*coordn);                                                       2649
 u = malloc(sizeof(double)*coordn);                                                         2650
 v = malloc(sizeof(double)*coordn);                                                         2651
 psi2 = malloc(sizeof(double)*2*(m+1)*(n+1));                                               2652
 u2 = malloc(sizeof(double)*2*(m+1)*(n+1));                                                 2653
 v2 = malloc(sizeof(double)*2*(m+1)*(n+1));                                                 2654
                                                                                            2655
                                                                                            2656
 getchebu1kcol2Dc (psik, uk, Lcheb, m, n);                                                  2657
 getfouru1krow2Dc (psik, vk, Lfour, m, n);                                                  2658
 for(i = 0; i<(m+1)*(n+1); i++)                                                             2659
        {vk[i][0] = (-1.0)*vk[i][0]; vk[i][1] = (-1.0)*vk[i][1];}                           2660
 if(!strcmp(valoutput,"YES"))                                                               2661
{ retphysmatpar(psi, coord, coordn, psik,Lfour, rank, numprocs, m, n, MPI_complex);         2662
 retphysmatpar(u, coord, coordn, uk,Lfour, rank, numprocs, m, n, MPI_complex);              2663
 retphysmatpar(v, coord, coordn, vk,Lfour, rank, numprocs, m, n, MPI_complex);              2664
}                                                                                           2665
                                                                                            2666
 for(i = 0; i<(m+1)*(n+1); i++)                                                             2667
        {psi2[2*i] = psik[i][0]; psi2[2*i+1] = psik[i][1];}                                 2668
 for(i = 0; i<(m+1)*(n+1); i++)                                                             2669
        {u2[2*i] = uk[i][0]; u2[2*i+1] = uk[i][1];}                                         2670
 for(i = 0; i<(m+1)*(n+1); i++)                                                             2671
        {v2[2*i] = vk[i][0]; v2[2*i+1] = vk[i][1];}                                         2672
                                                                                            2673
                                                                                            2674
if (!strcmp(option, "ASCII") || !strcmp(option, "BOTH"))                                    2675
        if(rank ==0)                                                                        2676
                {                                                                           2677
                 if(!strcmp(valoutput,"YES"))                                               2678
                 {strcpy(outstr1,outstr);                                                   2679
                 strcat(outstr1,"_valN");                                                   2680
                 num2str((*outputcount),outstr2);//printf("\noutstr2 = %s \n",outstr2    2681
                     );
                 strcat(outstr1,outstr2);                                                   2682
                 strcat(outstr1,".m");                                                      2683
//               printf("\noutstr : %s\n",outstr1);                                        2684
                                                                                            2685
                outputTfp = fopen(outstr1,"w");                                             2686
                fprintf(outputTfp,"\npsi = "); fprint2Dmat2(outputTfp,psi,coordy,           2687
```

```
                    coordx);
                fprintf(outputTfp,"\nu = "); fprint2Dmat2(outputTfp,u,coordy,coordx);
                fprintf(outputTfp,"\nv = "); fprint2Dmat2(outputTfp,v,coordy,coordx);

                fclose(outputTfp);
                 }

                 strcpy(outstr1,outstr);
                 strcat(outstr1,"_coeffsN");
                 num2str((*outputcount),outstr2);//printf("\noutstr2 = %s \n",outstr2
                     );
                 strcat(outstr1,outstr2);
                 strcat(outstr1,".txt");
//               printf("\noutstr : %s\n",outstr1);

                 outputTfp = fopen(outstr1,"w");
                 fprintf(outputTfp,"\npsik = "); fprint2Dc2(outputTfp,psik,m+1,n+1);
                 fprintf(outputTfp,"\nuk = "); fprint2Dc2(outputTfp,uk,m+1,n+1);
                 fprintf(outputTfp,"\nvk = "); fprint2Dc2(outputTfp,vk,m+1,n+1);
                fclose(outputTfp);
                }

if (!strcmp(option, "BIN") || !strcmp(option, "BOTH"))
        if(rank ==0)
                {
                 if(!strcmp(valoutput,"YES"))
                 {strcpy(outstr1,outstr);
                 strcat(outstr1,"_valN");
                 num2str((*outputcount),outstr2);//printf("\noutstr2 = %s \n",outstr2
                     );
                 strcat(outstr1,outstr2);
                 strcat(outstr1,".bin");
//               printf("\noutstr : %s\n",outstr1);

                 outputTfp = fopen(outstr1,"wb");

                fwrite(psi, sizeof(double), coordy*coordx, outputTfp);
                fwrite(u, sizeof(double), coordy*coordx, outputTfp);
                fwrite(v, sizeof(double), coordy*coordx, outputTfp);
                fclose(outputTfp);
                }

                 strcpy(outstr1,outstr);
                 strcat(outstr1,"_coeffsN");
                 num2str((*outputcount),outstr2);//printf("\noutstr2 = %s \n",outstr2
                     );
                 strcat(outstr1,outstr2);
                 strcat(outstr1,".bin");
//               printf("\noutstr : %s\n",outstr1);

                 outputTfp2 = fopen(outstr1,"wb");

                fwrite(psi2, sizeof(double), 2*(m+1)*(n+1), outputTfp2);
                fwrite(u2, sizeof(double), 2*(m+1)*(n+1), outputTfp2);
                fwrite(v2, sizeof(double), 2*(m+1)*(n+1), outputTfp2);

                fclose(outputTfp2);
                }
        (*outputcount)=(*outputcount)+1;

if(rank == 0)
        {strcpy(outstr1,outstr);
        strcat(outstr1,"_TIME.txt");
        outputTfp = fopen(outstr1,"a");
```

```
        if((*eventflag)) fprintf(outputTfp,"\nEvent Has Ocurred                          2748
            !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n");
        fprintf(outputTfp,"\n%g \t\t %16.16g \t\t %16.16g [s]", (*outputcount)-1.0,     2749
            nondimtime,dimtime);
        fclose(outputTfp);                                                             2750
        }                                                                              2751
        (*eventflag) = 0;                                                              2752
                                                                                       2753
 fftw_free(uk);                                                                        2754
 fftw_free(vk);                                                                        2755
 free(psi);                                                                            2756
 free(u);                                                                              2757
 free(v);                                                                              2758
 free(psi2);                                                                           2759
 free(u2);                                                                             2760
 free(v2);                                                                             2761
 return 1;                                                                             2762
                                                                                       2763
}                                                                                      2764
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++      2765
                                                                                       2766
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++      2767
int printpsiuv(fftw_complex* psik, double *outputcount, double outputsperfile, char*   2768
    outstr, char *option, char *valoutput, int coordx, int coordy, long unsigned int
    coordn, double *coord,double Lcheb, double Lfour, int m, int n, int rank, int
    numprocs, int* eventflag, double nondimtime, double dimtime, MPI_Datatype
    MPI_complex)
{char outstr1[STRMAX*5], outstr2[STRMAX*5];                                            2769
 FILE *outputTfp,*outputTfp2;                                                          2770
 fftw_complex *uk, *vk;                                                                2771
 double *psi, *u, *v,*psi2,*u2,*v2;                                                    2772
 int i,j;                                                                              2773
 static double filecount = 0.0;                                                        2774
 static double intrafilecount = 0.0;                                                   2775
                                                                                       2776
 uk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));                    2777
 vk = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));                    2778
 psi = malloc(sizeof(double)*coordn);                                                  2779
 u = malloc(sizeof(double)*coordn);                                                    2780
 v = malloc(sizeof(double)*coordn);                                                    2781
 psi2 = malloc(sizeof(double)*2*(m+1)*(n+1));                                          2782
 u2 = malloc(sizeof(double)*2*(m+1)*(n+1));                                            2783
 v2 = malloc(sizeof(double)*2*(m+1)*(n+1));                                            2784
                                                                                       2785
 if(((*outputcount)/outputsperfile == (filecount + 1)) || ((*outputcount)/             2786
     outputsperfile > (filecount + 1)))
 {filecount = filecount+1.0;                                                           2787
 intrafilecount = 0.0;                                                                 2788
 }                                                                                     2789
                                                                                       2790
                                                                                       2791
 getchebu1kcol2Dc (psik, uk, Lcheb, m, n);                                             2792
 getfouru1krow2Dc (psik, vk, Lfour, m, n);                                             2793
 for(i = 0; i<(m+1)*(n+1); i++)                                                        2794
        {vk[i][0] = (-1.0)*vk[i][0]; vk[i][1] = (-1.0)*vk[i][1];}                      2795
 if(!strcmp(valoutput,"YES"))                                                          2796
{ retphysmatpar(psi, coord, coordn, psik,Lfour, rank, numprocs, m, n, MPI_complex);    2797
 retphysmatpar(u, coord, coordn, uk,Lfour, rank, numprocs, m, n, MPI_complex);         2798
 retphysmatpar(v, coord, coordn, vk,Lfour, rank, numprocs, m, n, MPI_complex);         2799
}                                                                                      2800
                                                                                       2801
 for(i = 0; i<(m+1)*(n+1); i++)                                                        2802
        {psi2[2*i] = psik[i][0]; psi2[2*i+1] = psik[i][1];}                            2803
 for(i = 0; i<(m+1)*(n+1); i++)                                                        2804
```

```
        {u2[2*i] = uk[i][0]; u2[2*i+1] = uk[i][1];}
 for(i = 0; i<(m+1)*(n+1); i++)
        {v2[2*i] = vk[i][0]; v2[2*i+1] = vk[i][1];}


if (!strcmp(option, "ASCII") || !strcmp(option, "BOTH"))
        if(rank ==0)
                {
                 if(!strcmp(valoutput,"YES"))
                 {strcpy(outstr1,outstr);
                 strcat(outstr1,"_valN");
                 num2str((filecount),outstr2);//printf("\noutstr2 = %s \n",outstr2);
                 strcat(outstr1,outstr2);
                 strcat(outstr1,".m");
//               printf("\noutstr : %s\n",outstr1);
                if(intrafilecount == 0.0)
                        outputTfp = fopen(outstr1,"w");
                else outputTfp = fopen(outstr1,"a");
                fprintf(outputTfp,"\npsi = "); fprint2Dmat2(outputTfp,psi,coordy,
                    coordx);
                fprintf(outputTfp,"\nu = "); fprint2Dmat2(outputTfp,u,coordy,coordx);
                fprintf(outputTfp,"\nv = "); fprint2Dmat2(outputTfp,v,coordy,coordx);

                fclose(outputTfp);
                 }

                 strcpy(outstr1,outstr);
                 strcat(outstr1,"_coeffsN");
                 num2str((filecount),outstr2);//printf("\noutstr2 = %s \n",outstr2);
                 strcat(outstr1,outstr2);
                 strcat(outstr1,".txt");
//               printf("\noutstr : %s\n",outstr1);

                if(intrafilecount == 0.0)
                        outputTfp = fopen(outstr1,"w");
                else outputTfp = fopen(outstr1,"a");

                 fprintf(outputTfp,"\npsik = "); fprint2Dc2(outputTfp,psik,m+1,n+1);
                 fprintf(outputTfp,"\nuk = "); fprint2Dc2(outputTfp,uk,m+1,n+1);
                 fprintf(outputTfp,"\nvk = "); fprint2Dc2(outputTfp,vk,m+1,n+1);
                fclose(outputTfp);
                }

if (!strcmp(option, "BIN") || !strcmp(option, "BOTH"))
        if(rank ==0)
                {

                 if(!strcmp(valoutput,"YES"))
                 {strcpy(outstr1,outstr);
                 strcat(outstr1,"_valN");
                 num2str((filecount),outstr2);//printf("\noutstr2 = %s \n",outstr2);
                 strcat(outstr1,outstr2);
                 strcat(outstr1,".bin");
//               printf("\noutstr : %s\n",outstr1);

                if(intrafilecount == 0.0)
                        {outputTfp = fopen(outstr1,"wb");}
                else {outputTfp = fopen(outstr1,"ab");}

                fwrite(psi, sizeof(double), coordy*coordx, outputTfp);
                fwrite(u, sizeof(double), coordy*coordx, outputTfp);
                fwrite(v, sizeof(double), coordy*coordx, outputTfp);
                fclose(outputTfp);
                }
```

```
                strcpy(outstr1,outstr);
                strcat(outstr1,"_coeffsN");
                num2str((filecount),outstr2);//printf("\noutstr2 = %s \n",outstr2);
                strcat(outstr1,outstr2);
                strcat(outstr1,".bin");
//              printf("\noutstr : %s\n",outstr1);

               if(intrafilecount == 0.0)
                       outputTfp2 = fopen(outstr1,"wb");
               else outputTfp2 = fopen(outstr1,"ab");

               fwrite(psi2, sizeof(double), 2*(m+1)*(n+1), outputTfp2);
               fwrite(u2, sizeof(double), 2*(m+1)*(n+1), outputTfp2);
               fwrite(v2, sizeof(double), 2*(m+1)*(n+1), outputTfp2);

               fclose(outputTfp2);
               }
        (*outputcount)=(*outputcount)+1;
        intrafilecount = intrafilecount +1;

if(rank == 0)
        {strcpy(outstr1,outstr);
        strcat(outstr1,"_TIME.txt");

        if((*outputcount) == 1.0)
                outputTfp = fopen(outstr1,"w");
        else
                outputTfp = fopen(outstr1,"a");

        if((*eventflag)) fprintf(outputTfp,"\nEvent Has Ocurred
            !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n");
        if(intrafilecount == 1.0 && outputsperfile > 1.0) fprintf(outputTfp,"\n\nFILE
             NO: %g\n", filecount);
        fprintf(outputTfp,"\n%g \t\t %g \t\t %g \t\t %16.16g \t\t %16.16g [s]", (*
            outputcount)-1.0,filecount, intrafilecount - 1, nondimtime,dimtime);
        fclose(outputTfp);
        }
        (*eventflag) = 0;

 fftw_free(uk);
 fftw_free(vk);
 free(psi);
 free(u);
 free(v);
 free(psi2);
 free(u2);
 free(v2);
 return 1;

}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++



//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Note:
// All values are in non-dimensionalised form
int tracker1(double *coord, double *coordn, fftw_complex *uk, fftw_complex *ukhalf,
    fftw_complex *ukn, fftw_complex *vk, fftw_complex *vkhalf, fftw_complex *vkn,
    double h,long unsigned int coordnum, double Lfour,int rank, int numprocs, int m,
    int n, MPI_Datatype MPI_complex)
{double *k1u, *k2u, *k3u,  *k4u,*k1v, *k2v, *k3v, *k4v, *coordint;
 int i,j,k;
```

```
                                                                                2926
 k1u = malloc(sizeof(double)*coordnum);                                         2927
 k2u = malloc(sizeof(double)*coordnum);                                         2928
 k3u = malloc(sizeof(double)*coordnum);                                         2929
 k4u = malloc(sizeof(double)*coordnum);                                         2930
 k1v = malloc(sizeof(double)*coordnum);                                         2931
 k2v = malloc(sizeof(double)*coordnum);                                         2932
 k3v = malloc(sizeof(double)*coordnum);                                         2933
 k4v = malloc(sizeof(double)*coordnum);                                         2934
 coordint = malloc(sizeof(double)*coordnum*2);                                  2935
                                                                                2936
 retphysmatpar(k1u, coordn, coordnum, ukn, Lfour,rank, numprocs, m, n, MPI_complex);  2937
 retphysmatpar(k1v, coordn, coordnum, vkn, Lfour,rank, numprocs, m, n, MPI_complex);  2938
                                                                                2939
 for(i = 0; i<coordnum; i++)                                                    2940
        {coordint[i*2] = coordn[i*2] + 0.5*h*k1u[i]; coordint[i*2+1] = coordn[i*2+1]  2941
            + 0.5*h*k1v[i];}
                                                                                2942
 retphysmatpar(k2u, coordint, coordnum, ukhalf, Lfour,rank, numprocs, m, n,     2943
     MPI_complex);
 retphysmatpar(k2v, coordint, coordnum, vkhalf, Lfour,rank, numprocs, m, n,     2944
     MPI_complex);
                                                                                2945
 for(i = 0; i<coordnum; i++)                                                    2946
        {coordint[i*2] = coordn[i*2] + 0.5*h*k2u[i]; coordint[i*2+1] = coordn[i*2+1]  2947
            + 0.5*h*k2v[i];}
                                                                                2948
 retphysmatpar(k3u, coordint, coordnum, ukhalf, Lfour,rank, numprocs, m, n,     2949
     MPI_complex);
 retphysmatpar(k3v, coordint, coordnum, vkhalf, Lfour,rank, numprocs, m, n,     2950
     MPI_complex);
                                                                                2951
 for(i = 0; i<coordnum; i++)                                                    2952
        {coordint[i*2] = coordn[i*2] + h*k3u[i]; coordint[i*2+1] = coordn[i*2+1] + h*  2953
            k3v[i];}
                                                                                2954
 retphysmatpar(k4u, coordint, coordnum, uk, Lfour,rank, numprocs, m, n, MPI_complex);  2955
 retphysmatpar(k4v, coordint, coordnum, vk, Lfour,rank, numprocs, m, n, MPI_complex);  2956
                                                                                2957
 for(i = 0; i<coordnum; i++)                                                    2958
        {coord[i*2] = coordn[i*2] + h/6.0 * (k1u[i] + 2.0*k2u[i] + 2.0*k3u[i] + k4u[i  2959
            ]); coord[i*2+1] = coordn[i*2+1] + h/6.0 * (k1v[i] + 2.0*k2v[i] + 2.0*k3v
            [i] + k4v[i]);}
                                                                                2960
                                                                                2961
 free(k1u);                                                                     2962
 free(k2u);                                                                     2963
 free(k3u);                                                                     2964
 free(k4u);                                                                     2965
 free(k1v);                                                                     2966
 free(k2v);                                                                     2967
 free(k3v);                                                                     2968
 free(k4v);                                                                     2969
 free(coordint);                                                                2970
                                                                                2971
 return 1;                                                                      2972
}                                                                               2973
                                                                                2974
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++   2975
                                                                                2976
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++   2977
// It is assumed that this function is called by root only                      2978
 readcoeffparams(char *instr, int* xres, int * yres, double *xlen, double *ylen,  2979
     double * U, double* outtime, double *time)
```

```
{char instr1[5*STRMAX];
 FILE *inputfp;

 strcpy(instr1, instr);
 strcat(instr1,".m");

 if(inputfp = fopen(instr1,"r"));else{printf("\n File not found: %s\n",instr1);return
      0;}

 printf("\nCoeffs Param file: %s\n",instr1);
 fnamesearch("TOTALTIME","=", inputfp);
 fgetval(time,inputfp);
 printf("\n Total Time = %g\n",(*time));


 fnamesearch("OUTPERIOD","=", inputfp);
 fgetval(outtime,inputfp);
 printf("\n Out Time = %g\n",(*outtime));

  fnamesearch("SIZE",":", inputfp);
 (*xres) = fgetdim(inputfp);
 (*yres) = fgetdim(inputfp);
 printf("\nxres: %d\t\tyres: %d\n",(*xres),(*yres));

 fnamesearch("LENGTHS",":", inputfp);
 fgetval(xlen,inputfp);
 fgetval(ylen,inputfp);
 printf("\nxlen: %16.16g\t\tylen: %16.16g\n",(*xlen),(*ylen));

 fnamesearch("UMAX","=", inputfp);
 fgetval(U,inputfp);
 printf("\nUmax = %16.16g\n",(*U));

 (*time) = (*time)*(*U)/(*ylen);
// (*outtime) = (*outtime)*(*U)/(*ylen);

 fclose(inputfp);

 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// It is assumed that this function is called by root only
int readcoord(char* coordinp, double **coord, long unsigned int *coordn)
{FILE * inputfp;

 if(inputfp = fopen(coordinp,"r"));else{printf("\n File not found: %s\n",coordinp);
     return 0;}
 fnamesearch("COORDN",":", inputfp);
 (*coordn) = fgetdim2(inputfp);
 printf("\ncoordnum = %lu\n",(*coordn));

 fnamesearch("COORD","=", inputfp);

if((*coord) = malloc(sizeof(double)*2*(*coordn)));else {printf("\nERROR\n");return
    0;}

 fgetmatrix((*coord),1, 2*(*coordn), inputfp);

 fclose(inputfp);

 return 1;
}
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int readcoeffs(char *instr, double *filecount,int increment, fftw_complex *uk,
    fftw_complex *vk, int m, int n, int rank, int numprocs)
{long unsigned int i;
//int i;
int j,retflag = 1;
 FILE *inputfp;
 char instr2[STRMAX*5], line[STRMAX];
 double *temp1;

 temp1 = malloc(sizeof(double)*(m+1)*(n+1)*3*2);


if(rank == 0)
{ strcpy(instr2, instr);
 strcat(instr2,"_coeffsN");
 num2str((*filecount),line);
 strcat(instr2,line);
 strcat(instr2,".bin");

// printf("\nCoeffs file: %s\n",instr2);
 if(inputfp = fopen(instr2,"rb"));else {printf("\nFile not found: %s\n",instr2);
     retflag = 0;}
 if(retflag == 1)
{ j = fread(temp1, sizeof(double), ((m+1)*(n+1)*2*3), inputfp);
// printf("\n%d items to be read, actual = %d\n", (m+1)*(n+1)*2*3,j);
 if(feof(inputfp) == 0);else printf("\nend of file reached!!\n");
 if(ferror(inputfp) == 0);else printf("\nError has occurred!!\n");
 fclose(inputfp);}
//print2D(temp1,2,(m+1)*(n+1)*3 );
 }

 MPI_Bcast(&retflag,1,MPI_INT,0,MPI_COMM_WORLD);
if(retflag == 0) return 0;
 MPI_Bcast(temp1,(m+1)*(n+1)*2*3,MPI_DOUBLE,0,MPI_COMM_WORLD);

/* for(i = 0;i<(m+1)*(n+1);i++)
        {uk[i][0] = temp1[i*2]; uk[i][1] = temp1[i*2+1];}
*/
 for(i = (m+1)*(n+1);i<2*(m+1)*(n+1);i++)
        {uk[i-(m+1)*(n+1)][0] = temp1[i*2]; uk[i-(m+1)*(n+1)][1] = temp1[i*2+1];}

 for(i = 2*(m+1)*(n+1);i<3*(m+1)*(n+1);i++)
        {vk[i - 2*(m+1)*(n+1)][0] = temp1[i*2]; vk[i - 2*(m+1)*(n+1)][1] = temp1[i
            *2+1];}

 (*filecount) =  (*filecount) + increment*1.0;

 free(temp1);

 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int printcoord(double *coord, double *outputcount, double outputsperfile, char*
    outstr, char *option, long unsigned int coordn,int rank, double nondimtime,
    double dimtime)
{char outstr1[STRMAX*5], outstr2[STRMAX*5];
 FILE *outputTfp;
 int i,j;
```

```
 static double filecount = 0.0;                                                                    3100
 static double intrafilecount = 0.0;                                                               3101
                                                                                                   3102
                                                                                                   3103
 if(((*outputcount)/outputsperfile == (filecount + 1)) || ((*outputcount)/                         3104
     outputsperfile > (filecount + 1)))
 {filecount = filecount+1;                                                                         3105
 intrafilecount = 0.0;                                                                             3106
 }                                                                                                 3107
                                                                                                   3108
                                                                                                   3109
if (!strcmp(option, "ASCII") || !strcmp(option, "BOTH"))                                           3110
        if(rank ==0)                                                                               3111
                {strcpy(outstr1,outstr);                                                           3112
                 strcat(outstr1,"_coordN");                                                        3113
                 num2str((filecount),outstr2);//printf("\noutstr2 = %s \n",outstr2);             3114
                 strcat(outstr1,outstr2);                                                          3115
                 strcat(outstr1,".m");                                                             3116
//               printf("\noutstr : %s\n",outstr1);                                               3117
                                                                                                   3118
                if(intrafilecount == 0.0)                                                          3119
                        outputTfp = fopen(outstr1,"w");                                            3120
                else outputTfp = fopen(outstr1,"a");                                               3121
                                                                                                   3122
                fprintf(outputTfp,"\ncoord = "); fprint2Dmat2(outputTfp,coord,coordn               3123
                    ,2);
                fclose(outputTfp);                                                                 3124
                }                                                                                  3125
                                                                                                   3126
if (!strcmp(option, "BIN") || !strcmp(option, "BOTH"))                                             3127
        if(rank ==0)                                                                               3128
                {strcpy(outstr1,outstr);                                                           3129
                 strcat(outstr1,"_coordN");                                                        3130
                 num2str((filecount),outstr2);//printf("\noutstr2 = %s \n",outstr2);             3131
                 strcat(outstr1,outstr2);                                                          3132
                 strcat(outstr1,".bin");                                                           3133
//               printf("\noutstr : %s\n",outstr1);                                               3134
                                                                                                   3135
                if(intrafilecount == 0.0)                                                          3136
                        outputTfp = fopen(outstr1,"wb");                                           3137
                else outputTfp = fopen(outstr1,"ab");                                              3138
                                                                                                   3139
                                                                                                   3140
                fwrite(coord, sizeof(double), coordn*2, outputTfp);                                3141
                fclose(outputTfp);                                                                 3142
                }                                                                                  3143
        (*outputcount)=(*outputcount)+1;                                                           3144
        intrafilecount = intrafilecount +1;                                                        3145
                                                                                                   3146
if(rank == 0)                                                                                      3147
        {strcpy(outstr1,outstr);                                                                   3148
        strcat(outstr1,"_coordTIME.txt");                                                          3149
//      outputTfp = fopen(outstr1,"a");                                                            3150
        //fprintf(outputTfp,"\n%g \t\t %16.16g \t\t %16.16g [s]", (*outputcount)-1.0,              3151
             nondimtime, dimtime);
        if((*outputcount) == 1.0)                                                                  3152
                outputTfp = fopen(outstr1,"w");                                                    3153
        else                                                                                       3154
                outputTfp = fopen(outstr1,"a");                                                    3155
                                                                                                   3156
        if(intrafilecount == 1.0 && outputsperfile > 1.0) fprintf(outputTfp,"\n\nFILE              3157
             NO: %g\n", filecount);
        fprintf(outputTfp,"\n%g \t\t %g \t\t %g \t\t %16.16g \t\t %16.16g [s]", (*                 3158
            outputcount)-1.0,filecount, intrafilecount - 1, nondimtime,dimtime);
```

```
                                                                                3159
                                                                                3160
        fclose(outputTfp);                                                      3161
        }                                                                       3162
                                                                                3163
if((*outputcount) == 2)                                                         3164
        if(rank == 0)                                                           3165
                {strcpy(outstr1,outstr);                                        3166
                strcat(outstr1,"coord.m");                                      3167
                outputTfp = fopen(outstr1,"w");                                 3168
                fprintf(outputTfp,"coordn = %lu;\ntstep = %16.16g;\n",coordn,dimtime)  3169
                    ;
                fclose(outputTfp);                                              3170
                }                                                               3171
                                                                                3172
 return 1;                                                                      3173
                                                                                3174
}                                                                               3175
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  3176
```

Low level sub-routines

```
//#include<cblas.h>                                                             1
#include<fftw3.h>                                                               2
#include<stdio.h>                                                               3
#include<stdlib.h>                                                              4
#include<stddef.h>                                                              5
#include<ctype.h>                                                               6
#include<math.h>                                                                7
#include<mpi.h>                                                                 8
#define STRMAX 50                                                               9
//#define SIZE 15                                                               10
#define PI 3.141592653589793                                                    11
//#define ITER 6000                                                             12
                                                                                13
                                                                                14
                                                                                15
                                                                                16
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  17
int ceil1(int n,int m)                                                          18
{double mdouble = m*1.0;                                                        19
 if((n/m) == (n)/(mdouble))                                                     20
   return (n/m);                                                                21
 else return (n/m +1);                                                          22
}                                                                               23
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  24
                                                                                25
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  26
int makezero1(double *u, int size)                                              27
{int i;                                                                         28
 for(i = 0;i<size;i++)                                                          29
        u[i] = 0.0;                                                             30
 return 1;                                                                      31
}                                                                               32
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  33
                                                                                34
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  35
int makezeroc(fftw_complex *u, int size)                                        36
{int i;                                                                         37
 for(i = 0;i<size;i++)                                                          38
        { u[i][0] = 0.0;u[i][1] = 0.0;}                                         39
 return 1;                                                                      40
}                                                                               41
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  42
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int makezeroimc(fftw_complex *u, int size)
{int i;
 for(i = 0;i<size;i++)
  u[i][1] = 0.0;
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int divc(fftw_complex quotient,fftw_complex divisor, fftw_complex *ans)//CHECKED
 {double conj;
  conj = divisor[0]*divisor[0]+divisor[1]*divisor[1];//printf("\nconj:%g\n",conj);
  //divisor[1] = divisor[1]*(-1.0);
  (*ans)[0] = (quotient[0]*divisor[0] + quotient[1]*divisor[1])/conj;
  (*ans)[1] = (-quotient[0]*divisor[1] + quotient[1]*divisor[0])/conj;
  return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int multc(fftw_complex a, fftw_complex b,fftw_complex *prod)//CHECKED
{
 (*prod)[0] = a[0]*b[0]-a[1]*b[1];
 (*prod)[1] = a[0]*b[1]+a[1]*b[0];
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int subtractc(fftw_complex a, fftw_complex b, fftw_complex *ans)//CHECKED
{
(*ans)[0] = a[0]-b[0];
(*ans)[1] = a[1] - b[1];
return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Does Gauss seidel elimination for either real or im part of complex nos
//rim = 0: real, rim = 1: imaginary
int matrixgausscrim_old(fftw_complex *A,fftw_complex *x, fftw_complex* b,int rim, int
     size)//CHECKED
{int i = 0;
 int j = 0;
 int k = 0;
 double temp = 0.0;

if(rim == 0)
{ for (i = 0; i<size; i++)
    { for (j = 0; j<size;j++)
          if((j !=i) && (A[j*size+i][0]!=0.0))
          {temp = A[j*size+i][0];
            b[j][0]=temp / A[i*size+i][0]*b[i][0] - b[j][0];
             for (k = 0; k<size;k++)
              {A[j*size+k][0] = temp / A[i*size+i][0] * A[i*size+k][0]-A[j*size+k
                  ][0];
              }
          }}
 for(i =0;i<size;i++)
     x[i][0]=b[i][0]/A[i*size+i][0];
}
else
```

```
{ for (i = 0; i<size; i++)
    { for (j = 0; j<size;j++)
          if((j !=i) && (A[j*size+i][1]!=0.0))
          {temp = A[j*size+i][1];
            b[j][1]=temp / A[i*size+i][1]*b[i][1] - b[j][1];
             for (k = 0; k<size;k++)
              {A[j*size+k][1] = temp / A[i*size+i][1] * A[i*size+k][1]-A[j*size+k
                  ][1];
              }
          }}
 for(i =0;i<size;i++)
     x[i][1]=b[i][1]/A[i*size+i][1];
}
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Does Gauss seidel elimination for either real or im part of complex nos
//rim = 0: real, rim = 1: imaginary
int matrixgausscrim(fftw_complex *A,fftw_complex *x, fftw_complex* b,int rim, int
    size)//CHECKED
{int i = 0;
 int j = 0;
 int k = 0;
 double temp = 0.0;
 int *ipiv;
 double *K;
 double *B;
 int size2 = size*size;
 int nrhs = 1,info,one = 1, two = 2;
 K = malloc(sizeof(double)*size*size);
 B = malloc(sizeof(double)*size);


ipiv = malloc(sizeof(int)*size);
char N = 'N';
if(rim == 0)
{//for(i = 0;i<size;i++)
 //     for(j = 0;j<size;j++)
 //             K[j*size+i] = A[i*size+j][0];

// for(i = 0;i<size;i++)
//      B[i] = b[i][0];

// dcopy_(&size2, &(A[0][0]), &two, K, &one);
 dcopy_(&size, &(b[0][0]), &two, B, &one);

//for(i = 0;i<size; i++)
//      for(j = 0;j<size;j++)
//              K[j*size+i] = A[i*size+j][0];
for(i = 0; i<size; i++)
        dcopy_(&size, &(A[i*size][0]),&two, &(K[i]), &size);

// clapack_dgesv(CblasColMajor,size,1,K,size,ipiv,B,size);
// dgesv_(&size,&nrhs,K,&size,ipiv,B,&size,&info);

  dgetrf_(&size,&size,K,&size,ipiv,&info);              /* factor A */
  dgetrs_(&N,&size,&nrhs,K,&size,ipiv,B,&size,&info);     /* solve for x */

// for(i = 0;i<size;i++)
//      x[i][0] = B[i][0] ;
 dcopy_(&size, B, &one, &(x[0][0]), &two);
}
```

```
else
{//for(i = 0;i<size;i++)
//      for(j = 0;j<size;j++)
//              K[j*size+i] = A[i*size+j][0];


// for(i = 0;i<size;i++)
//      B[i] = b[i][1];
// dcopy_(&size2, &(A[0][1]), &two, K, &one);
 dcopy_(&size, &(b[0][1]), &two, B, &one);

//for(i = 0;i<size; i++)
//      for(j = 0;j<size;j++)
//              K[j*size+i] = A[i*size+j][1];
for(i = 0; i<size; i++)
        dcopy_(&size, &(A[i*size][1]),&two, &(K[i]), &size);

// clapack_dgesv(CblasColMajor,size,1,K,size,ipiv,B,size);
// dgesv_(&size,&nrhs,K,&size,ipiv,B,&size,&info);
  dgetrf_(&size,&size,K,&size,ipiv,&info);              /* factor A */
  dgetrs_(&N,&size,&nrhs,K,&size,ipiv,B,&size,&info);     /* solve for x */

// for(i = 0;i<size;i++)
//      x[i][1] = B[i][1] ;
 dcopy_(&size, B, &one, &(x[0][1]), &two);
}

 free(K);
 free(B);
 free(ipiv);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Does Gauss seidel elimination for either real or im part of complex nos
//rim = 0: real, rim = 1: imaginary
int matrixgausscrim_td(fftw_complex *A,fftw_complex *x, fftw_complex* b,int rim, int
    size)//CHECKED
{int i = 0;
 int j = 0;
 int k = 0;
 double temp = 0.0;
 int *ipiv;
 double *K;
 double *B;
 int size2 = size*size;
 int nrhs = 1,info,one = 1, two = 2;
 double *X, *Y, *ksi, *eta, E, H;


// K = malloc(sizeof(double)*size*size);
// B = malloc(sizeof(double)*size);
 X = malloc(sizeof(double)*size);
 Y = malloc(sizeof(double)*size);
 ksi = malloc(sizeof(double)*size);
 eta = malloc(sizeof(double)*size);

//ipiv = malloc(sizeof(int)*size);
char N = 'N';
if(rim == 0)
{
```

```
 X[size-1] = 0.0;
 Y[size-1] = 0.0;
 eta[0] = 0.0;
 ksi[0] = 1.0;

 X[size-2] = (-1.0)*A[(size-1)*size+size-2][0]/A[(size-1)*size+size-1][0];
 Y[size-2] = b[size-1][0]/A[(size-1)*size+size-1][0] ;

 for (i = size-2;i>0;i--)
        {X[i-1] = -A[i*size+i-1][0]/(A[i*size+i][0] + A[i*size+i+1][0]*X[i]);
        Y[i-1] = (b[i][0] - A[i*size+i+1][0]*Y[i])/(A[i*size+i][0] + A[i*size+i
            +1][0]*X[i]);
        }

 for (i = 1; i<size; i++)
        {ksi[i] = X[i-1]*ksi[i-1];
        eta[i] = X[i-1]*eta[i-1] + Y[i-1];
        }

 H = 0.0;
 E = A[0][0];

 for(i = 1;i<size;i++)
        {H += A[i][0]*eta[i];
        E += A[i][0]*ksi[i];
        }

 x[0][0] = (-H + b[0][0])/E;

for(i = 1; i<size; i++)
        {x[i][0] = ksi[i]*x[0][0] +  eta[i];}

}

else
{
// dcopy_(&size, &(b[0][1]), &two, B, &one);

//for(i = 0; i<size; i++)
//      dcopy_(&size, &(A[i*size][1]),&two, &(K[i]), &size);

 X[size-1] = 0.0;
 Y[size-1] = 0.0;
 eta[0] = 0.0;
 ksi[0] = 1.0;

 X[size-2] = -A[(size-1)*size+size-2][1]/A[(size-1)*size+size-1][1];
 Y[size-2] = b[size-1][1]/A[(size-1)*size+size-1][1] ;

 for (i = size-2;i>0;i--)
        {X[i-1] = -A[i*size+i-1][1]/(A[i*size+i][1] + A[i*size+i+1][1]*X[i]);
        Y[i-1] = (b[i][1] - A[i*size+i+1][1]*Y[i])/(A[i*size+i][1] + A[i*size+i
            +1][1]*X[i]);
        }

 for (i = 1; i<size; i++)
        {ksi[i] = X[i-1]*ksi[i-1];
        eta[i] = X[i-1]*eta[i-1] + Y[i-1];
        }

 H = 0.0;
 E = A[0][1];
 for(i = 1;i<size;i++)
```

```
        {H += A[i][1]*eta[i];
        E += A[i][1]*ksi[i];
        }

 x[0][1] = (-H + b[0][1])/E;

for(i = 1; i<size; i++)
        {x[i][1] = ksi[i]*x[0][1] +  eta[i];}

//  dgetrf_(&size,&size,K,&size,ipiv,&info);              /* factor A */
//  dgetrs_(&N,&size,&nrhs,K,&size,ipiv,B,&size,&info);     /* solve for x */

// dcopy_(&size, B, &one, &(x[0][1]), &two);
}

// free(K);
// free(B);
// free(ipiv);

 free(X);
 free(Y);
 free(ksi);
 free(eta);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
/*//Does Gauss seidel elimination for complex nos //CHECKED
int matrixgaussc(fftw_complex *A,fftw_complex *x, fftw_complex* b, int size)
{int i = 0;
 int j = 0;
 int k = 0;
 fftw_complex temp;
 temp[0] = temp[1] = 0;
 fftw_complex div,sub,prod;
 for (i = 0; i<size; i++)
    { for (j = 0; j<size;j++)
          if((j !=i) && (A[j*size+i][0]!=0.0))
          {temp[0] = A[j*size+i][0];temp[1] = A[j*size+i][1];
           divc(temp,A[i*size+i],&div);
           multc(div,b[i],&prod);
           subtractc(prod,b[j],&sub);
           b[j][0] = sub[0]; b[j][1] = sub[1];
             for (k = 0; k<size;k++)
              {divc(temp, A[i*size+i],&div);
               multc(div, A[i*size+k],&prod);
               subtractc(prod,A[j*size+k],&sub);
               A[j*size+k][0]= sub[0];A[j*size+k][1]= sub[1];
              }
          }}
 for(i =0;i<size;i++)
    { divc(b[i],A[i*size+i],&div);x[i][0]=div[0];x[i][1]=div[1];}
 return 1;
}*/
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Assembles the chebyshev Matrix //not needed
int assembleT(double *T, int n)
{int i,j;
 for(i = 0;i<n+1;i++)
 {for(j = 0;j<n+1;j++)
```

```
 {if(j == 0 ) T[i*(n+1)+j] = 1.0;                                                    356
 else if(j ==1) T[i*(n+1)+j] = -cos(PI*i/n);                                         357
 else T[i*(n+1)+j] = 2*-cos(PI*i/n)*T[i*(n+1)+j-1]- T[i*(n+1)+j-2]  ;                358
 }                                                                                   359
 }                                                                                   360
 return 1;                                                                           361
}                                                                                    362
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++    363
                                                                                     364
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++    365
int copycomplex(fftw_complex *source, fftw_complex *dest,int size)                   366
{int k;                                                                              367
 int one = 1;                                                                        368
// for(k = 0;k<size;k++)                                                            369
// {dest[k][0] = source[k][0];                                                      370
// dest[k][1] = source[k][1];                                                       371
// }                                                                                372
 zcopy_(&size, source, &one, dest, &one);                                            373
 return 1;                                                                           374
}                                                                                    375
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++    376
                                                                                     377
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++    378
int addcomplex(fftw_complex *source, fftw_complex *dest,int size)                    379
{int k;                                                                              380
 int one = 1; double alpha = 1.0;                                                    381
// for(k = 0;k<size;k++)                                                            382
// {dest[k][0] += source[k][0];                                                     383
// dest[k][1] += source[k][1];                                                      384
// }                                                                                385
 zaxpy_(&size, &alpha, source, &one, dest, &one);                                    386
 return 1;                                                                           387
}                                                                                    388
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++    389
                                                                                     390
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++    391
int dotmultcrim(fftw_complex *a, fftw_complex *b,fftw_complex *c,int size)           392
{int k;                                                                              393
 for(k = 0;k<size;k++)                                                               394
 {c[k][0] = a[k][0]*b[k][0];                                                         395
 c[k][1] = a[k][1]*b[k][1];                                                          396
 }                                                                                   397
 return 1;                                                                           398
}                                                                                    399
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++    400
                                                                                     401
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++    402
int dotmultr(double *a, double *b, double *c, int n)                                 403
{int k;                                                                              404
 for(k = 0;k<n+1;k++)                                                                405
     c[k] = a[k]*b[k];                                                               406
 return 1;                                                                           407
}                                                                                    408
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++    409
                                                                                     410
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++    411
int matrixmultc1 (fftw_complex *A, fftw_complex *B,fftw_complex *C,int m,int n,int p)  412
    //CHECKED
{int i, j, k;                                                                        413
 for (i=0;i<m;i++)                                                                   414
     {for (j=0;j<p;j++)                                                              415
         {C[i*p+j][0]=0.0;                                                           416
         for(k =0;k<n;k++)                                                           417
             {C[i*p+j][0]+=A[i*n+k][0]*B[k*p+j][0];}                                 418
```

```
         }                                                                                  419
     }                                                                                      420
 for (i=0;i<m;i++)                                                                          421
     {for (j=0;j<p;j++)                                                                     422
         {C[i*p+j][1]=0.0;                                                                  423
         for(k =0;k<n;k++)                                                                  424
             {C[i*p+j][1]+=A[i*n+k][1]*B[k*p+j][1];}                                        425
         }                                                                                  426
     }                                                                                      427
return 1;                                                                                   428
}                                                                                           429
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  430
                                                                                            431
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  432
int print2Dc(fftw_complex *matrix,int m,int n )                                             433
{int i = 0;                                                                                 434
 int j = 0;double a,b;                                                                      435
 for (i=0;i<m;i++)                                                                          436
     for(j=0;j<n;j++)                                                                       437
     {//if(matrix[i*n+j][0]<1e-17&&matrix[i*n+j][0]>-1e-17) a = 0.0; else a = matrix[       438
         i*n+j][0];
      //if(matrix[i*n+j][1]<1e-17&&matrix[i*n+j][1]>-1e-17) b = 0.0; else b = matrix[       439
          i*n+j][1];
      printf("%1.5g+%1.5gi\t",matrix[i*n+j][0],matrix[i*n+j][1]);                           440
     if(j ==n-1)printf("\n");                                                               441
     }                                                                                      442
 return 1;                                                                                  443
 }                                                                                          444
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  445
                                                                                            446
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  447
int print2Dc2(fftw_complex *matrix,int m,int n )                                            448
{int i = 0;                                                                                 449
 int j = 0;double a,b;                                                                      450
 for (i=0;i<m;i++)                                                                          451
     for(j=0;j<n;j++)                                                                       452
     {//if(matrix[i*n+j][0]<1e-17&&matrix[i*n+j][0]>-1e-17) a = 0.0; else a = matrix[       453
         i*n+j][0];
      //if(matrix[i*n+j][1]<1e-17&&matrix[i*n+j][1]>-1e-17) b = 0.0; else b = matrix[       454
          i*n+j][1];
      printf("%16.16g %16.16gi\t",matrix[i*n+j][0],matrix[i*n+j][1]);                       455
     if(j ==n-1)printf("\n");                                                               456
     }                                                                                      457
 return 1;                                                                                  458
 }                                                                                          459
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  460
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  461
int fprint2Dc(FILE*fp,fftw_complex *matrix,int m,int n )                                    462
{int i = 0;                                                                                 463
 int j = 0;double a,b;                                                                      464
 for (i=0;i<m;i++)                                                                          465
     for(j=0;j<n;j++)                                                                       466
     {//if(matrix[i*n+j][0]<1e-17&&matrix[i*n+j][0]>-1e-17) a = 0.0; else a = matrix[       467
         i*n+j][0];
      //if(matrix[i*n+j][1]<1e-17&&matrix[i*n+j][1]>-1e-17) b = 0.0; else b = matrix[       468
          i*n+j][1];
      fprintf(fp,"%8.8g+%8.8gi\t",matrix[i*n+j][0],matrix[i*n+j][1]);                       469
     if(j ==n-1)fprintf(fp,"\n");                                                           470
     }                                                                                      471
 return 1;                                                                                  472
 }                                                                                          473
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  474
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  475
int fprint2Dc2(FILE*fp,fftw_complex *matrix,int m,int n )                                   476
```

```
{int i = 0;
 int j = 0;double a,b;
 for (i=0;i<m;i++)
     for(j=0;j<n;j++)
     {//if(matrix[i*n+j][0]<1e-17&&matrix[i*n+j][0]>-1e-17) a = 0.0; else a = matrix[
         i*n+j][0];
      //if(matrix[i*n+j][1]<1e-17&&matrix[i*n+j][1]>-1e-17) b = 0.0; else b = matrix[
          i*n+j][1];
      fprintf(fp,"%16.16g %16.16gi\t",matrix[i*n+j][0],matrix[i*n+j][1]);
     //if(j ==n-1)fprintf(fp,"\n");
     }
printf("\n");
 return 1;
 }
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int print2Dmatc(fftw_complex *matrix,int m,int n )
{int i = 0;
 int j = 0;double a,b;
 printf("[ ");
 for (i=0;i<m;i++)
     for(j=0;j<n;j++)
     {//if(matrix[i*n+j][0]<1e-17&&matrix[i*n+j][0]>-1e-17) a = 0.0; else a = matrix[
         i*n+j][0];
      //if(matrix[i*n+j][1]<1e-17&&matrix[i*n+j][1]>-1e-17) b = 0.0; else b = matrix[
          i*n+j][1];
      printf("%8.8g+%8.8gi, ",matrix[i*n+j][0],matrix[i*n+j][1]);
     if(j ==n-1)printf(";");
     }printf("];\n");
 return 1;
 }

int fprint2Dmatc(FILE*fp,fftw_complex *matrix,int m,int n )
{int i = 0;
 int j = 0;double a,b;
 fprintf(fp,"[ ");
 for (i=0;i<m;i++)
     for(j=0;j<n;j++)
     {//if(matrix[i*n+j][0]<1e-17&&matrix[i*n+j][0]>-1e-17) a = 0.0; else a = matrix[
         i*n+j][0];
      //if(matrix[i*n+j][1]<1e-17&&matrix[i*n+j][1]>-1e-17) b = 0.0; else b = matrix[
          i*n+j][1];
      fprintf(fp,"%8.8g+%8.8gi, ",matrix[i*n+j][0],matrix[i*n+j][1]);
     if(j ==n-1)fprintf(fp,";");
     }fprintf(fp,"];\n");
 return 1;
 }
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int print2Dmatc2(fftw_complex *matrix,int m,int n )
{int i = 0;
 int j = 0;double a,b;
 printf("[ ");
 for (i=0;i<m;i++)
     for(j=0;j<n;j++)
     {//if(matrix[i*n+j][0]<1e-17&&matrix[i*n+j][0]>-1e-17) a = 0.0; else a = matrix[
         i*n+j][0];
      //if(matrix[i*n+j][1]<1e-17&&matrix[i*n+j][1]>-1e-17) b = 0.0; else b = matrix[
          i*n+j][1];
      printf("%16.16g+%16.16gi, ",matrix[i*n+j][0],matrix[i*n+j][1]);
     if(j ==n-1)printf(";");
     }printf("];\n");
 return 1;
```

```
 }
int fprint2Dmatc2(FILE*fp,fftw_complex *matrix,int m,int n )
{int i = 0;
 int j = 0;double a,b;
 fprintf(fp,"[ ");
 for (i=0;i<m;i++)
     for(j=0;j<n;j++)
     {//if(matrix[i*n+j][0]<1e-17&&matrix[i*n+j][0]>-1e-17) a = 0.0; else a = matrix[
         i*n+j][0];
      //if(matrix[i*n+j][1]<1e-17&&matrix[i*n+j][1]>-1e-17) b = 0.0; else b = matrix[
          i*n+j][1];
      fprintf(fp,"%16.16g+%16.16gi, ",matrix[i*n+j][0],matrix[i*n+j][1]);
     if(j ==n-1)fprintf(fp,";");
     }fprintf(fp,"];\n");
 return 1;
 }
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Calculates the Cheb coeffs when the cheb approx is colwise
int chebforcol2Dc(fftw_complex *u, fftw_complex *uk, fftw_complex *incheb,
    fftw_complex *outcheb, fftw_plan pchebfor, int m, int n)
{int i, j, colno; double ck;
 int size = m+1, one = 1, min1 = -1, nplus1 = n+1,mmin1 = m-1; double alpha = 1.0/m;
 makezeroc(uk, (m+1)*(n+1));
 for (colno = 0; colno<n+1;colno++)
    {//for(i = 0;i<m+1;i++)
      //   {incheb[m-i][0] = u[i*(n+1)+colno][0];incheb[m-i][1] = u[i*(n+1)+colno
          ][1];}//
        zcopy_(&size, &(u[colno]), &nplus1, &(incheb[0]), &min1);

//     for(i = m+1;i<2*m;i++)
//         {incheb[i][0] = incheb[2*m-i][0]; incheb[i][1] = incheb[2*m-i][1];}
        zcopy_(&mmin1, &(incheb[1]),&min1, &(incheb[m+1]),&one);
     fftw_execute(pchebfor);
//     for(i = 0;i<m+1;i++)
//         {if(i==0||i ==m) ck =2.0;else ck =1.0; uk[i*(n+1)+colno][0] = outcheb[i
    ][0]/m/ck; uk[i*(n+1)+colno][1] = outcheb[i][1]/m/ck;}
        zcopy_(&size, outcheb, &one, &(uk[colno]),&nplus1);
        zscal_(&size, &alpha, &(uk[colno]),&nplus1);
        uk[colno][0] = uk[colno][0]/2.0; uk[colno][1] = uk[colno][1]/2.0; uk[m*(n+1)+
            colno][0] = uk[m*(n+1)+colno][0]/2.0; uk[m*(n+1)+colno][1] = uk[m*(n+1)+
            colno][1]/2.0;
     }

 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Calculates the Cheb coeffs when the cheb approx is colwise
int chebforcol2Dcpar(fftw_complex *u, fftw_complex *uk, fftw_complex *incheb,
    fftw_complex *outcheb, fftw_plan pchebfor, int m, int n, MPI_Datatype MPI_complex
    )
{int i, j, colno,rowno; double ck;
 int n1 = -1 ,n2 = -1, *tasks = NULL, *tasksnum = NULL, numtasks = -1, numprocs = -1,
     rank = -1;
 int size = m+1, one = 1, min1 = -1, nplus1 = n+1, size2,mmin1 = m-1; double alpha =
     1.0/m;
 fftw_complex *utemp;
 int *recvcounts, * displs;

  int blockcounts[1];
```

```
  MPI_Status  Stat;
/*  MPI_Datatype MPI_complex,oldtypes[1];
  MPI_Aint offsets[1];
 offsets[0] = 0;
 blockcounts[0] = 2;
 oldtypes[0] = MPI_DOUBLE;
 MPI_Type_struct(1,blockcounts,offsets,oldtypes,&MPI_complex);
 MPI_Type_commit (&MPI_complex);
*/
        { MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
          MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        }
// if(numprocs>8) numprocs = 8;

 tasks = malloc(sizeof(int)*numprocs);
 tasksnum = malloc(sizeof(int)*numprocs);
 utemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));

/* for(colno = 0;colno<n+1;colno++)
        for(rowno = 0; rowno < m+1; rowno++)
                {utemp[colno*(m+1)+ rowno][0] = u[rowno*(n+1) + colno][0]; utemp[
                    colno*(m+1)+ rowno][1] = u[rowno*(n+1) + colno][1]; }
*/

 getn1n2(&n1, &n2, rank, tasks, tasksnum, &numtasks, numprocs, n+1);
 size2 = (m+1)*(n2-n1+1);
 recvcounts = malloc(sizeof(int)*numprocs);
 displs = malloc(sizeof(int)*numprocs);
 for(i = 0,j=0;i<numprocs;i++)
        {recvcounts[i] = tasksnum[i] * (m+1);
         displs[i] = j;
        if(tasks[i]!= -1)
                j=j+recvcounts[i];
        }

// makezeroc(uk, (m+1)*(n+1));
if(tasks[rank]!=-1)
 for (rowno = n1; rowno<n2+1;rowno++)
    {//for(i = 0;i<m+1;i++)
     //    {incheb[m-i][0] = utemp[rowno*(m+1)+i][0];incheb[m-i][1] = utemp[rowno*(m
         +1)+i][1];}//
        zcopy_(&size, &(u[rowno]), &nplus1, &(incheb[0]), &min1);
//     for(i = m+1;i<2*m;i++)
//         {incheb[i][0] = incheb[2*m-i][0]; incheb[i][1] = incheb[2*m-i][1];}
        zcopy_(&mmin1, &(incheb[1]),&min1, &(incheb[m+1]),&one);
     fftw_execute(pchebfor);
//     for(i = 0;i<m+1;i++)
//         {if(i==0||i ==m) ck =2.0;else ck =1.0; uk[rowno*(m+1)+i][0] = outcheb[i
    ][0]/m/ck; uk[rowno*(m+1)+i][1] = outcheb[i][1]/m/ck;}
        zcopy_(&size, outcheb, &one, &(uk[rowno*(m+1)]), &one);
        zscal_(&size, &alpha, &(uk[rowno*(m+1)]),&one);
        uk[rowno*(m+1)][0] = uk[rowno*(m+1)][0]/2.0; uk[rowno*(m+1)][1] = uk[rowno*(m
            +1)][1]/2.0; uk[rowno*(m+1)+m][0] = uk[rowno*(m+1)+m][0]/2.0; uk[rowno*(m
            +1)+m][1] = uk[rowno*(m+1)+m][1]/2.0;
     }
// MPI_Barrier(MPI_COMM_WORLD);
 if(rank<numtasks && rank !=0)
        MPI_Send(&(uk[n1*(m+1)]), (m+1)*(n2-n1+1), MPI_complex, 0, rank,
            MPI_COMM_WORLD);

if(rank == 0)
        {for(i = 1;i<numtasks;i++)
                MPI_Recv(&(utemp[tasks[i]*(m+1)]), (m+1)*tasksnum[i], MPI_complex, i,
                     i, MPI_COMM_WORLD, &Stat);
```

```
//      for(i = 0; i <(m+1)*(n2-n1+1);i++)
//              {utemp[i][0] = uk[i][0]; utemp[i][1] = uk[i][1];}
        zcopy_(&size2, uk, &one, utemp, &one);
        }



//MPI_Gatherv(&(uk[n1*(m+1)]), (m+1)*(n2-n1+1), MPI_complex, utemp, recvcounts,
    displs, MPI_complex, 0, MPI_COMM_WORLD);

//for(colno = 0; colno<n+1; colno++)
//      for(rowno = 0; rowno<m+1; rowno++)
//              {uk[rowno*(n+1) + colno][0] = utemp[colno*(m+1) + rowno][0]; uk[rowno
    *(n+1) + colno][1] = utemp[colno*(m+1) + rowno][1];}
 for(rowno = 0; rowno<n+1;rowno++)
        zcopy_(&size, &(utemp[rowno*(m+1)]), &one, &(uk[rowno]), &nplus1);

//// MPI_Barrier(MPI_COMM_WORLD);
 MPI_Bcast(uk,(m+1)*(n+1),MPI_complex,0,MPI_COMM_WORLD);

fftw_free(utemp);
free(tasks);
free(tasksnum);
free(recvcounts);
free(displs);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Calculates the Cheb coeffs when the cheb approx is colwise
int chebforcol2Dcpar2(fftw_complex *u, fftw_complex *uk, fftw_complex *incheb,
    fftw_complex *outcheb, fftw_plan pchebfor, int m, int n, MPI_Comm communicator,
    int numprocs, int rank, MPI_Datatype MPI_complex)
{int i, j, colno,rowno; double ck;
 int n1 = -1 ,n2 = -1, *tasks = NULL, *tasksnum = NULL, numtasks = -1;// numprocs =
     -1,rank = -1;
 int size = m+1, one = 1, min1 = -1, nplus1 = n+1, size2,mmin1 = m-1; double alpha =
     1.0/m;
 fftw_complex *utemp;
 int *recvcounts, * displs;

//  int blockcounts[1];
  MPI_Status Stat;
/*  MPI_Datatype MPI_complex,oldtypes[1];
  MPI_Aint offsets[1];
 offsets[0] = 0;
 blockcounts[0] = 2;
 oldtypes[0] = MPI_DOUBLE;
 MPI_Type_struct(1,blockcounts,offsets,oldtypes,&MPI_complex);
 MPI_Type_commit (&MPI_complex);
*/
//      { MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
//       MPI_Comm_rank(MPI_COMM_WORLD, &rank);
//      }
// if(numprocs>8) numprocs = 8;

 tasks = malloc(sizeof(int)*numprocs);
 tasksnum = malloc(sizeof(int)*numprocs);
 utemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));

/* for(colno = 0;colno<n+1;colno++)
        for(rowno = 0; rowno < m+1; rowno++)
                {utemp[colno*(m+1)+ rowno][0] = u[rowno*(n+1) + colno][0]; utemp[
```

```
                    colno*(m+1)+ rowno][1] = u[rowno*(n+1) + colno][1]; }
*/
 getn1n2(&n1, &n2, rank, tasks, tasksnum, &numtasks, numprocs, n+1);
 size2 = (m+1)*(n2-n1+1);
 recvcounts = malloc(sizeof(int)*numprocs);
 displs = malloc(sizeof(int)*numprocs);
 for(i = 0,j=0;i<numprocs;i++)
        {recvcounts[i] = tasksnum[i] * (m+1);
         displs[i] = j;
        if(tasks[i]!= -1)
                j=j+recvcounts[i];
        }

// makezeroc(uk, (m+1)*(n+1));
if(tasks[rank]!=-1)
 for (rowno = n1; rowno<n2+1;rowno++)
    {//for(i = 0;i<m+1;i++)
     //    {incheb[m-i][0] = utemp[rowno*(m+1)+i][0];incheb[m-i][1] = utemp[rowno*(m
         +1)+i][1];}//
        zcopy_(&size, &(u[rowno]), &nplus1, &(incheb[0]), &min1);
//     for(i = m+1;i<2*m;i++)
//         {incheb[i][0] = incheb[2*m-i][0]; incheb[i][1] = incheb[2*m-i][1];}
        zcopy_(&mmin1, &(incheb[1]),&min1, &(incheb[m+1]),&one);
     fftw_execute(pchebfor);
//     for(i = 0;i<m+1;i++)
//         {if(i==0||i ==m) ck =2.0;else ck =1.0; uk[rowno*(m+1)+i][0] = outcheb[i
    ][0]/m/ck; uk[rowno*(m+1)+i][1] = outcheb[i][1]/m/ck;}
        zcopy_(&size, outcheb, &one, &(uk[rowno*(m+1)]), &one);
        zscal_(&size, &alpha, &(uk[rowno*(m+1)]),&one);
        uk[rowno*(m+1)][0] = uk[rowno*(m+1)][0]/2.0; uk[rowno*(m+1)][1] = uk[rowno*(m
            +1)][1]/2.0; uk[rowno*(m+1)+m][0] = uk[rowno*(m+1)+m][0]/2.0; uk[rowno*(m
            +1)+m][1] = uk[rowno*(m+1)+m][1]/2.0;
     }
// MPI_Barrier(communicator);
 if(rank<numtasks && rank !=0)
        MPI_Send(&(uk[n1*(m+1)]), (m+1)*(n2-n1+1), MPI_complex, 0, rank, communicator
            );

if(rank == 0)
        {for(i = 1;i<numtasks;i++)
                MPI_Recv(&(utemp[tasks[i]*(m+1)]), (m+1)*tasksnum[i], MPI_complex, i,
                     i, communicator, &Stat);
//      for(i = 0; i <(m+1)*(n2-n1+1);i++)
//              {utemp[i][0] = uk[i][0]; utemp[i][1] = uk[i][1];}
        zcopy_(&size2, uk, &one, utemp, &one);
        }



//MPI_Gatherv(&(uk[n1*(m+1)]), (m+1)*(n2-n1+1), MPI_complex, utemp, recvcounts,
    displs, MPI_complex, 0, communicator);

//for(colno = 0; colno<n+1; colno++)
//      for(rowno = 0; rowno<m+1; rowno++)
//              {uk[rowno*(n+1) + colno][0] = utemp[colno*(m+1) + rowno][0]; uk[rowno
    *(n+1) + colno][1] = utemp[colno*(m+1) + rowno][1];}
 for(rowno = 0; rowno<n+1;rowno++)
        zcopy_(&size, &(utemp[rowno*(m+1)]), &one, &(uk[rowno]), &nplus1);

//// MPI_Barrier(communicator);
 MPI_Bcast(uk,(m+1)*(n+1),MPI_complex,0,communicator);

fftw_free(utemp);
```

```
free(tasks);
free(tasksnum);
free(recvcounts);
free(displs);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Calculates the physical values from the Cheb coeffs whrn the cheb approx is colwise
int chebbackcol2Dc(fftw_complex *u, fftw_complex *uk, fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebback, int m, int n, MPI_Datatype MPI_complex)
{int i, j, colno;double ck;
int mplus1 = m+1, one = 1, min1 = -1, nplus1 = n+1, size2,mmin1 = m-1; double alpha = 1.0/m;
 makezeroc(uk, (m+1)*(n+1));
 for (colno = 0; colno<n+1;colno++)
    {for(i = 0;i<m+1;i++)
         {if(i==0||i==m) ck = 1.0;else ck = 2.0; incheb[i][0] = u[(i)*(n+1)+colno][0]/ck; incheb[i][1] = u[(i)*(n+1)+colno][1]/ck;}
     for(i = m+1;i<2*m;i++)
         {incheb[i][0] = incheb[2*m-i][0];incheb[i][1] = incheb[2*m-i][1];}

     fftw_execute(pchebback);
     for(i = 0;i<m+1;i++)
         {uk[(m-i)*(n+1)+colno][0] = outcheb[i][0];uk[(m-i)*(n+1)+colno][1] = outcheb[i][1];}//
     }

  return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Calculates the physical values from the Cheb coeffs whrn the cheb approx is colwise
int chebbackcol2Dcpar(fftw_complex *u, fftw_complex *uk, fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebback, int m, int n, MPI_Datatype MPI_complex)
{int i, j, colno,rowno;double ck;
 int n1 = -1 ,n2 = -1, *tasks = NULL, *tasksnum = NULL, numtasks = -1, numprocs = -1, rank = -1;
 int size = m+1, one = 1, min1 = -1, nplus1 = n+1, size2,mmin1 = m-1,mmin2 = m-2, sizemin2 = m-1; double alpha = 1.0/m, oneby2 = 0.5;

 fftw_complex *utemp;
 int *recvcounts, * displs;

  MPI_Status Stat;

        { MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
         MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        }

 tasks = malloc(sizeof(int)*numprocs);
 tasksnum = malloc(sizeof(int)*numprocs);
 utemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));

// for(colno = 0;colno<n+1;colno++)
//      for(rowno = 0; rowno < m+1; rowno++)
//              {utemp[colno*(m+1)+ rowno][0] = u[rowno*(n+1) + colno][0]; utemp[
```

```
    colno*(m+1)+ rowno][1] = u[rowno*(n+1) + colno][1]; }
 for(colno = 0; colno<n+1;colno++)
        zcopy_(&size, &(u[colno]), &nplus1, &(utemp[colno*(m+1)]), &one);

 getn1n2(&n1, &n2, rank, tasks, tasksnum, &numtasks, numprocs, n+1);
 size2 = (m+1)*(n2-n1+1);
 recvcounts = malloc(sizeof(int)*numprocs);
 displs = malloc(sizeof(int)*numprocs);
 for(i = 0,j=0;i<numprocs;i++)
        {recvcounts[i] = tasksnum[i] * (m+1);
         displs[i] = j;
        if(tasks[i]!= -1)
                j=j+recvcounts[i];
        }

// makezeroc(uk, (m+1)*(n+1));
if(tasks[rank]!=-1)
 for (rowno = n1; rowno< n2+1;rowno++)
    {for(i = 0;i<m+1;i++)
         {if(i==0||i==m) ck = 1.0;else ck = 2.0; incheb[i][0] = utemp[rowno*(m+1)+i
             ][0]/ck; incheb[i][1] = utemp[rowno*(m+1)+i][1]/ck
             ;}////////////////////////////////////////////////////////
//      zcopy_(&size, &(u[rowno]), &nplus1, incheb,&one);
//      zscal_(&mmin1, &oneby2, &(incheb[1]), &one);
//      incheb[0][0] = incheb[0][0]*2.0; incheb[0][1] = incheb[0][1]*2.0; incheb[m
    ][0] = incheb[m][0]*2.0; incheb[m][1] = incheb[m][1]*2.0;
     //for(i = m+1;i<2*m;i++)
     //    {incheb[i][0] = incheb[2*m-i][0];incheb[i][1] = incheb[2*m-i][1];}
        zcopy_(&mmin1, &(incheb[1]),&min1, &(incheb[m+1]),&one);
     fftw_execute(pchebback);
     //for(i = 0;i<m+1;i++)
     //    {uk[rowno*(m+1)+(m-i)][0] = outcheb[i][0];uk[rowno*(m+1)+(m-i)][1] =
          outcheb[i][1];}//
        zcopy_(&size, outcheb, &one, &(uk[rowno*(m+1)]), &min1);
     }

// MPI_Barrier(MPI_COMM_WORLD);
if(rank<numtasks && rank !=0)
 MPI_Send(&(uk[n1*(m+1)]), (m+1)*(n2-n1+1), MPI_complex, 0, rank, MPI_COMM_WORLD);

if(rank == 0)
        {for(i = 1;i<numtasks;i++)
                MPI_Recv(&(utemp[tasks[i]*(m+1)]), (m+1)*tasksnum[i], MPI_complex, i,
                     i, MPI_COMM_WORLD, &Stat);
        //for(i = 0; i <(m+1)*(n2-n1+1);i++)
        //      {utemp[i][0] = uk[i][0]; utemp[i][1] = uk[i][1];}
        zcopy_(&size2, uk, &one, utemp, &one);
        }

//MPI_Gatherv(&(uk[n1*(m+1)]), (m+1)*(n2-n1+1), MPI_complex, utemp, recvcounts,
    displs, MPI_complex, 0, MPI_COMM_WORLD);

//for(colno = 0; colno<n+1; colno++)
//      for(rowno = 0; rowno<m+1; rowno++)
//              {uk[rowno*(n+1) + colno][0] = utemp[colno*(m+1) + rowno][0]; uk[rowno
    *(n+1) + colno][1] = utemp[colno*(m+1) + rowno][1];}

 for(rowno = 0; rowno<n+1;rowno++)
        zcopy_(&size, &(utemp[rowno*(m+1)]), &one, &(uk[rowno]), &nplus1);

//// MPI_Barrier(MPI_COMM_WORLD);
 MPI_Bcast(uk,(m+1)*(n+1),MPI_complex,0,MPI_COMM_WORLD);

fftw_free(utemp);
```

```
free(tasks);
free(tasksnum);
free(recvcounts);
free(displs);
  return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Calculates the physical values from the Cheb coeffs whrn the cheb approx is colwise
int chebbackcol2Dcpar2(fftw_complex *u, fftw_complex *uk, fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebback, int m, int n, MPI_Comm communicator, int numprocs, int rank, MPI_Datatype MPI_complex)
{int i, j, colno,rowno;double ck;
 int n1 = -1 ,n2 = -1, *tasks = NULL, *tasksnum = NULL, numtasks = -1;//, numprocs = -1,rank = -1;
 int size = m+1, one = 1, min1 = -1, nplus1 = n+1, size2,mmin1 = m-1,mmin2 = m-2, sizemin2 = m-1; double alpha = 1.0/m, oneby2 = 0.5;

 fftw_complex *utemp;
 int *recvcounts, * displs;

//  int blockcounts[1];
  MPI_Status Stat;
/*  MPI_Datatype MPI_complex,oldtypes[1];
  MPI_Aint offsets[1];
 offsets[0] = 0;
 blockcounts[0] = 2;
 oldtypes[0] = MPI_DOUBLE;
 MPI_Type_struct(1,blockcounts,offsets,oldtypes,&MPI_complex);
 MPI_Type_commit (&MPI_complex);
*/
//      { MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
//       MPI_Comm_rank(MPI_COMM_WORLD, &rank);
//      }

 tasks = malloc(sizeof(int)*numprocs);
 tasksnum = malloc(sizeof(int)*numprocs);
 utemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));

// for(colno = 0;colno<n+1;colno++)
//      for(rowno = 0; rowno < m+1; rowno++)
//              {utemp[colno*(m+1)+ rowno][0] = u[rowno*(n+1) + colno][0]; utemp[colno*(m+1)+ rowno][1] = u[rowno*(n+1) + colno][1]; }
 for(colno = 0; colno<n+1;colno++)
        zcopy_(&size, &(u[colno]), &nplus1, &(utemp[colno*(m+1)]), &one);

 getn1n2(&n1, &n2, rank, tasks, tasksnum, &numtasks, numprocs, n+1);
 size2 = (m+1)*(n2-n1+1);
 recvcounts = malloc(sizeof(int)*numprocs);
 displs = malloc(sizeof(int)*numprocs);
 for(i = 0,j=0;i<numprocs;i++)
        {recvcounts[i] = tasksnum[i] * (m+1);
         displs[i] = j;
        if(tasks[i]!= -1)
                j=j+recvcounts[i];
        }

// makezeroc(uk, (m+1)*(n+1));
if(tasks[rank]!=-1)
 for (rowno = n1; rowno< n2+1;rowno++)
    {for(i = 0;i<m+1;i++)
         {if(i==0||i==m) ck = 1.0;else ck = 2.0; incheb[i][0] = utemp[rowno*(m+1)+i
```

```
            ][0]/ck; incheb[i][1] = utemp[rowno*(m+1)+i][1]/ck
            ;}////////////////////////////////////////////////////////
//      zcopy_(&size, &(u[rowno]), &nplus1, incheb,&one);                                923
//      zscal_(&mmin1, &oneby2, &(incheb[1]), &one);                                     924
//      incheb[0][0] = incheb[0][0]*2.0; incheb[0][1] = incheb[0][1]*2.0; incheb[m      925
    ][0] = incheb[m][0]*2.0; incheb[m][1] = incheb[m][1]*2.0;
     //for(i = m+1;i<2*m;i++)                                                          926
     //    {incheb[i][0] = incheb[2*m-i][0];incheb[i][1] = incheb[2*m-i][1];}          927
        zcopy_(&mmin1, &(incheb[1]),&min1, &(incheb[m+1]),&one);                       928
     fftw_execute(pchebback);                                                          929
     //for(i = 0;i<m+1;i++)                                                            930
     //    {uk[rowno*(m+1)+(m-i)][0] = outcheb[i][0];uk[rowno*(m+1)+(m-i)][1] =        931
         outcheb[i][1];}//
        zcopy_(&size, outcheb, &one, &(uk[rowno*(m+1)]), &min1);                       932
     }                                                                                 933
                                                                                       934
// MPI_Barrier(communicator);                                                         935
if(rank<numtasks && rank !=0)                                                          936
 MPI_Send(&(uk[n1*(m+1)]), (m+1)*(n2-n1+1), MPI_complex, 0, rank, communicator);       937
                                                                                       938
if(rank == 0)                                                                         939
        {for(i = 1;i<numtasks;i++)                                                     940
                MPI_Recv(&(utemp[tasks[i]*(m+1)]), (m+1)*tasksnum[i], MPI_complex, i,  941
                     i, communicator, &Stat);
        //for(i = 0; i <(m+1)*(n2-n1+1);i++)                                           942
        //      {utemp[i][0] = uk[i][0]; utemp[i][1] = uk[i][1];}                      943
        zcopy_(&size2, uk, &one, utemp, &one);                                         944
        }                                                                              945
                                                                                       946
//MPI_Gatherv(&(uk[n1*(m+1)]), (m+1)*(n2-n1+1), MPI_complex, utemp, recvcounts,        947
    displs, MPI_complex, 0, communicator);
                                                                                       948
//for(colno = 0; colno<n+1; colno++)                                                   949
//      for(rowno = 0; rowno<m+1; rowno++)                                             950
//              {uk[rowno*(n+1) + colno][0] = utemp[colno*(m+1) + rowno][0]; uk[rowno   951
    *(n+1) + colno][1] = utemp[colno*(m+1) + rowno][1];}
                                                                                       952
 for(rowno = 0; rowno<n+1;rowno++)                                                     953
        zcopy_(&size, &(utemp[rowno*(m+1)]), &one, &(uk[rowno]), &nplus1);             954
                                                                                       955
//// MPI_Barrier(communicator);                                                        956
 MPI_Bcast(uk,(m+1)*(n+1),MPI_complex,0,communicator);                                 957
                                                                                       958
fftw_free(utemp);                                                                      959
free(tasks);                                                                           960
free(tasksnum);                                                                        961
free(recvcounts);                                                                      962
free(displs);                                                                          963
  return 1;                                                                            964
}                                                                                      965
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++      966
                                                                                       967
                                                                                       968
                                                                                       969
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++      970
//Calculates the fourier coefficents (forward) where the fourier approx is Columnwise  971
int fourforcol2Dc (fftw_complex *u, fftw_complex *uk, fftw_complex *in, fftw_complex   972
    *out, fftw_plan pfor, int m, int n, MPI_Datatype MPI_complex)
{int i, colno;                                                                         973
 for(colno = 0;colno<n+1;colno++)                                                      974
     {for(i = 0;i<m+1;i++)                                                             975
         {in[i][0] = u[i*(n+1)+colno][0];                                              976
          in[i][1] = u[i*(n+1)+colno][1];                                              977
         }                                                                             978
```

```
      fftw_execute(pfor);
      for(i = 0;i<m+1;i++)
         {uk[i*(n+1)+colno][0] = out[i][0]/(n+1);
          uk[i*(n+1)+colno][1] = out[i][1]/(n+1);
         }
     }
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Calculates the fourier coefficents (forward) where the fourier approx is rowwise
int fourforrow2Dc (fftw_complex *u, fftw_complex *uk, fftw_complex *in, fftw_complex
    *out, fftw_plan pfor, int m, int n, MPI_Datatype MPI_complex)
{int i, rowno;
 for(rowno = 0;rowno<m+1;rowno++)
     {for(i = 0;i<n+1;i++)
         {in[i][0] = u[rowno*(n+1)+i][0];
          in[i][1] = u[rowno*(n+1)+i][1];
         }
      fftw_execute(pfor);
      for(i = 0;i<n+1;i++)
         {uk[rowno*(n+1)+i][0] = out[i][0]/(n+1);///
          uk[rowno*(n+1)+i][1] = out[i][1]/(n+1);///
         }
     }
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Calculates the fourier coefficents (forward) where the fourier approx is rowwise
int fourforrow2Dcpar (fftw_complex *u, fftw_complex *uk, fftw_complex *in,
    fftw_complex *out, fftw_plan pfor, int m, int n, MPI_Datatype MPI_complex)
{int i,j,colno, rowno;
 int *recvcounts, * displs;
fftw_complex *utemp;
 int n1 = -1 ,n2 = -1, *tasks = NULL, *tasksnum = NULL, numtasks = -1, numprocs = -1,
     rank = -1;
 int size = n+1, one = 1, min1 = -1, size2; double alpha = 1.0/(n+1);

//  int blockcounts[1];
  MPI_Status Stat;
/*  MPI_Datatype MPI_complex,oldtypes[1];
  MPI_Aint offsets[1];
 offsets[0] = 0;
 blockcounts[0] = 2;
 oldtypes[0] = MPI_DOUBLE;
 MPI_Type_struct(1,blockcounts,offsets,oldtypes,&MPI_complex);
 MPI_Type_commit (&MPI_complex);
*/
utemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
// if(numprocs == -1 || rank ==-1)
        { MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
          MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        }
// if(numprocs>8) numprocs = 8;

// if (tasks == NULL)
 tasks = malloc(sizeof(int)*numprocs);
// if (tasksnum == NULL)
tasksnum = malloc(sizeof(int)*numprocs);

//if(n1 == -1 || n2 == -1)
 getn1n2(&n1, &n2, rank, tasks, tasksnum, &numtasks, numprocs, m+1);
 size2 = (n+1)*(n2-n1+1);
```

```
 recvcounts = malloc(sizeof(int)*numprocs);
 displs = malloc(sizeof(int)*numprocs);
 for(i = 0,j=0;i<numprocs;i++)
        {recvcounts[i] = tasksnum[i] * (n+1);
         displs[i] = j;
        if(tasks[i]!= -1)
                j=j+recvcounts[i];
        }

if(tasks[rank]!=-1)
 for(rowno = n1;rowno<n2+1;rowno++)
     {//for(i = 0;i<n+1;i++)
      //   {in[i][0] = u[rowno*(n+1)+i][0]; in[i][1] = u[rowno*(n+1)+i][1];}
        zcopy_(&size, &(u[rowno*(n+1)]), &one, in, &one);
      fftw_execute(pfor);
      //for(i = 0;i<n+1;i++)
      //   {utemp[rowno*(n+1)+i][0] = out[i][0]/(n+1); utemp[rowno*(n+1)+i][1] = out[
          i][1]/(n+1);}
        zcopy_(&size, out, &one, &(utemp[rowno*(n+1)]), &one);
        zscal_(&size, &alpha, &(utemp[rowno*(n+1)]), &one);
     }
//MPI_Barrier(MPI_COMM_WORLD);
if(rank<numtasks && rank !=0)
 MPI_Send(&(utemp[n1*(n+1)]), (n+1)*(n2-n1+1), MPI_complex, 0, rank, MPI_COMM_WORLD);

if(rank == 0)
        {for(i = 1;i<numtasks;i++)
                MPI_Recv(&(uk[tasks[i]*(n+1)]), (n+1)*tasksnum[i], MPI_complex, i, i,
                     MPI_COMM_WORLD, &Stat);
        //for(i = 0; i <(n+1)*(n2-n1+1);i++)
        //      {uk[i][0] = utemp[i][0]; uk[i][1] = utemp[i][1];}
        zcopy_(&size2, utemp, &one, uk, &one);
        }

// MPI_Gatherv(&(utemp[n1*(n+1)]), (n+1)*(n2-n1+1), MPI_complex, uk, recvcounts,
    displs, MPI_complex, 0, MPI_COMM_WORLD);

//// MPI_Barrier(MPI_COMM_WORLD);
 MPI_Bcast(uk,(m+1)*(n+1),MPI_complex,0,MPI_COMM_WORLD);

fftw_free(utemp);
free(tasks);
free(tasksnum);
free(recvcounts);
free(displs);

}
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Calculates the fourier coefficents (forward) where the fourier approx is rowwise
int fourforrow2Dcpar2(fftw_complex *u, fftw_complex *uk, fftw_complex *in,
    fftw_complex *out, fftw_plan pfor, int m, int n, MPI_Comm communicator, int
    numprocs, int rank, MPI_Datatype MPI_complex)
{int i,j,colno, rowno;
 int *recvcounts, * displs;
fftw_complex *utemp;
 int n1 = -1 ,n2 = -1, *tasks = NULL, *tasksnum = NULL, numtasks = -1;//, numprocs =
     -1,rank = -1;
 int size = n+1, one = 1, min1 = -1, size2; double alpha = 1.0/(n+1);

//  int blockcounts[1];
  MPI_Status Stat;
/*  MPI_Datatype MPI_complex,oldtypes[1];
```

```
  MPI_Aint offsets[1];
 offsets[0] = 0;
 blockcounts[0] = 2;
 oldtypes[0] = MPI_DOUBLE;
 MPI_Type_struct(1,blockcounts,offsets,oldtypes,&MPI_complex);
 MPI_Type_commit (&MPI_complex);
*/
utemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
// if(numprocs == -1 || rank ==-1)
//       { MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
//        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
//       }
// if(numprocs>8) numprocs = 8;

// if (tasks == NULL)
 tasks = malloc(sizeof(int)*numprocs);
// if (tasksnum == NULL)
tasksnum = malloc(sizeof(int)*numprocs);

//if(n1 == -1 || n2 == -1)
 getn1n2(&n1, &n2, rank, tasks, tasksnum, &numtasks, numprocs, m+1);
 size2 = (n+1)*(n2-n1+1);
 recvcounts = malloc(sizeof(int)*numprocs);
 displs = malloc(sizeof(int)*numprocs);
 for(i = 0,j=0;i<numprocs;i++)
        {recvcounts[i] = tasksnum[i] * (n+1);
         displs[i] = j;
        if(tasks[i]!= -1)
                j=j+recvcounts[i];
        }

if(tasks[rank]!=-1)
 for(rowno = n1;rowno<n2+1;rowno++)
     {//for(i = 0;i<n+1;i++)
      //   {in[i][0] = u[rowno*(n+1)+i][0]; in[i][1] = u[rowno*(n+1)+i][1];}
        zcopy_(&size, &(u[rowno*(n+1)]), &one, in, &one);
      fftw_execute(pfor);
      //for(i = 0;i<n+1;i++)
      //   {utemp[rowno*(n+1)+i][0] = out[i][0]/(n+1); utemp[rowno*(n+1)+i][1] = out[
          i][1]/(n+1);}
        zcopy_(&size, out, &one, &(utemp[rowno*(n+1)]), &one);
        zscal_(&size, &alpha, &(utemp[rowno*(n+1)]), &one);
     }
//MPI_Barrier(communicator);
if(rank<numtasks && rank !=0)
 MPI_Send(&(utemp[n1*(n+1)]), (n+1)*(n2-n1+1), MPI_complex, 0, rank, communicator);

if(rank == 0)
        {for(i = 1;i<numtasks;i++)
                MPI_Recv(&(uk[tasks[i]*(n+1)]), (n+1)*tasksnum[i], MPI_complex, i, i,
                     communicator, &Stat);
        //for(i = 0; i <(n+1)*(n2-n1+1);i++)
        //      {uk[i][0] = utemp[i][0]; uk[i][1] = utemp[i][1];}
        zcopy_(&size2, utemp, &one, uk, &one);
        }

// MPI_Gatherv(&(utemp[n1*(n+1)]), (n+1)*(n2-n1+1), MPI_complex, uk, recvcounts,
    displs, MPI_complex, 0, communicator);

//// MPI_Barrier(communicator);
 MPI_Bcast(uk,(m+1)*(n+1),MPI_complex,0,communicator);

fftw_free(utemp);
free(tasks);
```

```
free(tasksnum);
free(recvcounts);
free(displs);

}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Calculates the physical value from fourier coeffecients (backward) where the fourier approx is Columnwise
int fourbackcol2Dc (fftw_complex *uk, fftw_complex *u, fftw_complex *in, fftw_complex *out, fftw_plan pback, int m, int n, MPI_Datatype MPI_complex)
{int i, colno;
 for(colno = 0;colno<n+1;colno++)
     {for(i = 0;i<m+1;i++)
         {in[i][0] = uk[i*(n+1)+colno][0];///(m+1);
          in[i][1] = uk[i*(n+1)+colno][1];///(m+1);
         }
      fftw_execute(pback);
      for(i = 0;i<m+1;i++)
         {u[i*(n+1)+colno][0] = out[i][0];
          u[i*(n+1)+colno][1] = out[i][1];
         }
     }
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Calculates the physical value from fourier coeffecients (backward) where the fourier approx is ROWwise
int fourbackrow2Dc (fftw_complex *uk, fftw_complex *u, fftw_complex *in, fftw_complex *out, fftw_plan pback, int m, int n, MPI_Datatype MPI_complex)
{int i, rowno;
 for(rowno = 0;rowno<m+1;rowno++)
     {for(i = 0;i<n+1;i++)
         {in[i][0] = uk[rowno*(n+1)+i][0];///(n+1);
          in[i][1] = uk[rowno*(n+1)+i][1];///(n+1);
         }
      fftw_execute(pback);
      for(i = 0;i<n+1;i++)
         {u[rowno*(n+1)+i][0] = out[i][0];
          u[rowno*(n+1)+i][1] = out[i][1];
         }
     }
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Calculates the physical value from fourier coeffecients (backward) where the fourier approx is ROWwise
int fourbackrow2Dcpar (fftw_complex *uk, fftw_complex *u, fftw_complex *in, fftw_complex *out, fftw_plan pback, int m, int n, MPI_Datatype MPI_complex)
{int i, rowno,colno,j;
 int *recvcounts, * displs;
 fftw_complex *utemp;
 int n1 = -1 ,n2 = -1, *tasks = NULL, *tasksnum = NULL, numtasks = -1, numprocs = -1, rank = -1;
 int size = n+1, one = 1, min1 = -1, size2; double alpha = 1.0/(n+1);

//  int blockcounts[1];
  MPI_Status Stat;
/*  MPI_Datatype MPI_complex,oldtypes[1];
  MPI_Aint offsets[1];
```

```
 offsets[0] = 0;
 blockcounts[0] = 2;
 oldtypes[0] = MPI_DOUBLE;
 MPI_Type_struct(1,blockcounts,offsets,oldtypes,&MPI_complex);
 MPI_Type_commit (&MPI_complex);
*/
utemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));

// if(numprocs == -1 || rank ==-1)
        { MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
         MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        }
// if(numprocs>8) numprocs = 8;

// if (tasks == NULL)
 tasks = malloc(sizeof(int)*numprocs);
// if (tasksnum == NULL)
tasksnum = malloc(sizeof(int)*numprocs);

//if(n1 == -1 || n2 == -1)
 getn1n2(&n1, &n2, rank, tasks, tasksnum, &numtasks, numprocs, m+1);
 size2 = (n+1)*(n2-n1+1);
 recvcounts = malloc(sizeof(int)*numprocs);
 displs = malloc(sizeof(int)*numprocs);
 for(i = 0,j=0;i<numprocs;i++)
        {recvcounts[i] = tasksnum[i] * (n+1);
         displs[i] = j;
        if(tasks[i]!= -1)
                j=j+recvcounts[i];
        }
if(tasks[rank]!=-1)
 for(rowno = n1;rowno<n2+1;rowno++)
     {//for(i = 0;i<n+1;i++)
      //   {in[i][0] = uk[rowno*(n+1)+i][0]; in[i][1] = uk[rowno*(n+1)+i][1];}
        zcopy_(&size, &(uk[rowno*(n+1)]), &one, in, &one);
       fftw_execute(pback);
       //for(i = 0;i<n+1;i++)
       //   {utemp[rowno*(n+1)+i][0] = out[i][0]; utemp[rowno*(n+1)+i][1] = out[i
           ][1];}
        zcopy_(&size, out, &one, &(utemp[rowno*(n+1)]), &one);
     }
//MPI_Barrier(MPI_COMM_WORLD);
if(rank<numtasks && rank !=0)
 MPI_Send(&(utemp[n1*(n+1)]), (n+1)*(n2-n1+1), MPI_complex, 0, rank, MPI_COMM_WORLD);

if(rank == 0)
        {for(i = 1;i<numtasks;i++)
                MPI_Recv(&(u[tasks[i]*(n+1)]), (n+1)*tasksnum[i], MPI_complex, i, i,
                    MPI_COMM_WORLD, &Stat);
        //for(i = 0; i <(n+1)*(n2-n1+1);i++)
        //      {u[i][0] = utemp[i][0]; u[i][1] = utemp[i][1];}
        zcopy_(&size2, utemp, &one, u, &one);
        }

// MPI_Gatherv(&(utemp[n1*(n+1)]), (n+1)*(n2-n1+1), MPI_complex, u, recvcounts,
    displs, MPI_complex, 0, MPI_COMM_WORLD);

//// MPI_Barrier(MPI_COMM_WORLD);
 MPI_Bcast(u,(m+1)*(n+1),MPI_complex,0,MPI_COMM_WORLD);

fftw_free(utemp);
free(tasks);
free(tasksnum);
free(recvcounts);
```

```
free(displs);

}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Calculates the physical value from fourier coeffecients (backward) where the fourier approx is ROWwise
int fourbackrow2Dcpar2(fftw_complex *uk, fftw_complex *u, fftw_complex *in, fftw_complex *out, fftw_plan pback, int m, int n, MPI_Comm communicator, int numprocs, int rank, MPI_Datatype MPI_complex)
{int i, rowno,colno,j;
 int *recvcounts, * displs;
 fftw_complex *utemp;
 int n1 = -1 ,n2 = -1, *tasks = NULL, *tasksnum = NULL, numtasks = -1;//, numprocs = -1,rank = -1;
 int size = n+1, one = 1, min1 = -1, size2; double alpha = 1.0/(n+1);

//  int blockcounts[1];
  MPI_Status Stat;
/*  MPI_Datatype MPI_complex,oldtypes[1];
  MPI_Aint offsets[1];
 offsets[0] = 0;
 blockcounts[0] = 2;
 oldtypes[0] = MPI_DOUBLE;
 MPI_Type_struct(1,blockcounts,offsets,oldtypes,&MPI_complex);
 MPI_Type_commit (&MPI_complex);
*/
utemp = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));

// if(numprocs == -1 || rank ==-1)
//       { MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
//        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
//       }
// if(numprocs>8) numprocs = 8;

// if (tasks == NULL)
 tasks = malloc(sizeof(int)*numprocs);
// if (tasksnum == NULL)
tasksnum = malloc(sizeof(int)*numprocs);

//if(n1 == -1 || n2 == -1)
 getn1n2(&n1, &n2, rank, tasks, tasksnum, &numtasks, numprocs, m+1);
 size2 = (n+1)*(n2-n1+1);
 recvcounts = malloc(sizeof(int)*numprocs);
 displs = malloc(sizeof(int)*numprocs);
 for(i = 0,j=0;i<numprocs;i++)
        {recvcounts[i] = tasksnum[i] * (n+1);
         displs[i] = j;
        if(tasks[i]!= -1)
                j=j+recvcounts[i];
        }
if(tasks[rank]!=-1)
 for(rowno = n1;rowno<n2+1;rowno++)
     {//for(i = 0;i<n+1;i++)
      //   {in[i][0] = uk[rowno*(n+1)+i][0]; in[i][1] = uk[rowno*(n+1)+i][1];}
        zcopy_(&size, &(uk[rowno*(n+1)]), &one, in, &one);
      fftw_execute(pback);
      //for(i = 0;i<n+1;i++)
      //   {utemp[rowno*(n+1)+i][0] = out[i][0]; utemp[rowno*(n+1)+i][1] = out[i][1];}
        zcopy_(&size, out, &one, &(utemp[rowno*(n+1)]), &one);
```

```
     }
//MPI_Barrier(communicator);
if(rank<numtasks && rank !=0)
 MPI_Send(&(utemp[n1*(n+1)]), (n+1)*(n2-n1+1), MPI_complex, 0, rank, communicator);

if(rank == 0)
        {for(i = 1;i<numtasks;i++)
                MPI_Recv(&(u[tasks[i]*(n+1)]), (n+1)*tasksnum[i], MPI_complex, i, i,
                    communicator, &Stat);
        //for(i = 0; i <(n+1)*(n2-n1+1);i++)
        //      {u[i][0] = utemp[i][0]; u[i][1] = utemp[i][1];}
        zcopy_(&size2, utemp, &one, u, &one);
        }

// MPI_Gatherv(&(utemp[n1*(n+1)]), (n+1)*(n2-n1+1), MPI_complex, u, recvcounts,
    displs, MPI_complex, 0, communicator);

//// MPI_Barrier(communicator);
 MPI_Bcast(u,(m+1)*(n+1),MPI_complex,0,communicator);

fftw_free(utemp);
free(tasks);
free(tasksnum);
free(recvcounts);
free(displs);


}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Gets 1st order derivatives of chebyshev coeffs approx for 2D where Chebyshev approx
     is columnwise
int getchebu1kcol2Dc (fftw_complex *uk, fftw_complex *u1k, double L, int m, int n)
{int colno,k,p;
 double ck;
 makezeroc(u1k,(n+1)*(m+1));
  for(colno = 0;colno<n+1;colno++)
   {for(k = 0;k<m-1+1;k++)
      {if(k==0) ck = 2.0; else ck = 1.0;
       for(p = k+1;p<m+1;p++)
           {if((p+k)/2.0 == (p+k)/2);
            else
               {u1k[k*(n+1)+colno][0] +=(2.0/L)*(2.0/ck)*p*uk[p*(n+1)+colno][0];u1k[k
                   *(n+1)+colno][1] +=(2.0/L)*(2.0/ck)*p*uk[p*(n+1)+colno][1];}
            }
       }
     u1k[m*(n+1)+colno][0]= u1k[m*(n+1)+colno][1] = 0.0;
   }
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Gets coeffs for 2nd derivatives of cheb coeffs when cheb approx is columnwise
int getchebu2kcol2Dc(fftw_complex *uk, fftw_complex *u2k,double L, int m, int n)
{int i,k,p;
double ck;
 makezeroc(u2k,(m+1)*(n+1));
for(i = 0;i<n+1;i++)
  {for(k = 0;k<m-2+1;k++)
     {if(k==0) ck = 2.0; else ck = 1.0;
     for(p = k+2;p<m+1;p++)
```

```
           if((p+k)/2 == (p+k)/2.0)                                                    1396
               {u2k[k*(n+1)+i][0] +=(2.0/L)*(2.0/L)*1.0/ck*p*(p*p-k*k)*uk[p*(n+1)+i    1397
                   ][0];
                u2k[k*(n+1)+i][0] +=(2.0/L)*(2.0/L)*1.0/ck*p*(p*p-k*k)*uk[p*(n+1)+i    1398
                    ][1];
                }                                                                      1399
     }                                                                                 1400
   u2k[(m-1)*(n+1)+i][0] = 0.0;                                                        1401
   u2k[m*(n+1)+i][0] = 0.0;                                                            1402
   u2k[(m-1)*(n+1)+i][1] = 0.0;                                                        1403
   u2k[m*(n+1)+i][1] = 0.0;                                                            1404
  }                                                                                    1405
 return 1;                                                                             1406
}                                                                                      1407
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++      1408
                                                                                       1409
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++      1410
//Gets coeffs for 2nd derivatives of fourier coeffs when fourier appprox is rowwise    1411
int getfouru2krow2Dc(fftw_complex *uk, fftw_complex *u2k, double Lfour,int m, int n)   1412
{int rowno, k,multiplier;                                                              1413
 for (rowno = 0;rowno<m+1; rowno++)                                                    1414
      {for(k = 0;k<n+1;k++)                                                            1415
           {if(k<ceil1((n+1),2)) multiplier = (2.0*PI/Lfour)*1.0*k;else multiplier =    1416
               (2.0*PI/Lfour)*1.0*(k-n-1);u2k[rowno*(n+1)+k][0] = -uk[rowno*(n+1)+k
               ][0] *(multiplier*1.0)*(multiplier*1.0);u2k[rowno*(n+1)+k][1] = -uk[
               rowno*(n+1)+k][1] *(multiplier*1.0)*(multiplier*1.0);}
      }                                                                                1417
}                                                                                      1418
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++      1419
                                                                                       1420
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++      1421
//Gets 1st order derivatives of Fourier coeffs for 2D where fourier approx is rowwise  1422
int getfouru1krow2Dc (fftw_complex *uk, fftw_complex *u1k, double Lfour,int m, int n)  1423
    //CHECKED
{int rowno, k,i;                                                                       1424
 fftw_complex multiplier,prod;                                                         1425
 for(rowno = 0; rowno<m+1; rowno++)                                                    1426
    {for(k = 0;k<n+1;k++)                                                              1427
        {multiplier[0] = 0.0; if(k <ceil1(n+1,2))multiplier[1] = (2.0*PI/Lfour)*1.0*k  1428
            ;else multiplier[1] =(2.0*PI/Lfour)*1.0*(k-n-1);
        multc(multiplier, uk[rowno*(n+1)+k],&prod);                                    1429
        u1k[rowno*(n+1)+k][0] =prod[0];                                                1430
        u1k[rowno*(n+1)+k][1] =prod[1];                                                1431
         }                                                                             1432
     }                                                                                 1433
        if((n+1)/2 ==(n+1)/2.0)                                                        1434
                for(i = 0;i<m+1;i++)                                                   1435
                        u1k[i*(n+1)+ceil1(n+1,2)][1] = 0.0;                            1436
                                                                                       1437
                for(i = 0;i<m+1;i++)                                                   1438
                        u1k[i*(n+1)][1] = 0.0;                                         1439
                                                                                       1440
 return 1;                                                                             1441
}                                                                                      1442
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++      1443
                                                                                       1444
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++      1445
//Gets coeffs for 2nd derivatives of cheb coeffs when cheb approx is rowwise          1446
int getchebu2krow2Dc(fftw_complex *uk, fftw_complex *u2k, int m, int n)                1447
{int i,k,p;                                                                            1448
double ck;                                                                             1449
 makezeroc(u2k,(m+1)*(n+1));                                                           1450
for(i = 0;i<m+1;i++)                                                                   1451
  {for(k = 0;k<n-2+1;k++)                                                              1452
```

```
     {if(k==0) ck = 2.0; else ck = 1.0;
      for(p = k+2;p<n+1;p++)
           if((p+k)/2 == (p+k)/2.0)
               {u2k[i*(n+1)+k][0] +=1.0/ck*p*(p*p-k*k)*uk[i*(n+1)+p][0];
                u2k[i*(n+1)+k][1] +=1.0/ck*p*(p*p-k*k)*uk[i*(n+1)+p][1];
                }
     }
   u2k[(i)*(n+1)+n-1][0] = 0.0;
   u2k[i*(n+1)+n][0] = 0.0;
   u2k[(i)*(n+1)+n-1][1] = 0.0;
   u2k[i*(n+1)+n][1] = 0.0;
  }
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Assembles the u velocity field for cheb coeffs arranged column wise
int calcu(fftw_complex *psik, fftw_complex *u, fftw_complex *in, fftw_complex *out,
    fftw_plan pback,fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebback,
     double Lcheb, int m, int n, MPI_Datatype MPI_complex)
{fftw_complex *tempu, *psi1k;
 int colno,rowno,i;
 tempu = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*(m+1));
 psi1k = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*(m+1));
 getchebu1kcol2Dc (psik, psi1k,Lcheb, m, n);

 //printf("\nCALCU: \n");///////////////////////
 //printf("\nPSI1K: \n");print2Dc(psi1k,m+1,n+1);/////////////////////////////////
 fourbackrow2Dc (psi1k, tempu, in, out, pback, m, n, MPI_complex);
 chebbackcol2Dc(tempu, u,incheb,outcheb,pchebback, m, n, MPI_complex);

// for(i = 0;i<(m+1)*(n+1);i++)
// {u[i][1] = 0.0;}

 fftw_free(tempu);
 fftw_free(psi1k);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Assembles the u velocity field for cheb coeffs arranged column wise
int calcupar(fftw_complex *psik, fftw_complex *u, fftw_complex *in, fftw_complex *out
    , fftw_plan pback,fftw_complex *incheb, fftw_complex *outcheb, fftw_plan
    pchebback, double Lcheb, int m, int n, MPI_Datatype MPI_complex)
{fftw_complex *tempu, *psi1k;
 int colno,rowno,i;
 tempu = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*(m+1));
 psi1k = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*(m+1));
 getchebu1kcol2Dc (psik, psi1k,Lcheb, m, n);

 //printf("\nCALCU: \n");///////////////////////
 //printf("\nPSI1K: \n");print2Dc(psi1k,m+1,n+1);/////////////////////////////////
 fourbackrow2Dcpar (psi1k, tempu, in, out, pback, m, n, MPI_complex);
 chebbackcol2Dcpar(tempu, u,incheb,outcheb,pchebback, m, n, MPI_complex);

// for(i = 0;i<(m+1)*(n+1);i++)
// {u[i][1] = 0.0;}

 fftw_free(tempu);
 fftw_free(psi1k);
 return 1;
}
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Assembles the u velocity field for cheb coeffs arranged column wise
int calcupar2(fftw_complex *psik, fftw_complex *u, fftw_complex *in, fftw_complex *
    out, fftw_plan pback,fftw_complex *incheb, fftw_complex *outcheb, fftw_plan
    pchebback, double Lcheb, int m, int n, MPI_Comm communicator, int numprocs, int
    rank, MPI_Datatype MPI_complex)
{fftw_complex *tempu, *psi1k;
 int colno,rowno,i;
 tempu = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*(m+1));
 psi1k = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*(m+1));
 getchebu1kcol2Dc (psik, psi1k,Lcheb, m, n);

 //printf("\nCALCU: \n");//////////////////////
 //printf("\nPSI1K: \n");print2Dc(psi1k,m+1,n+1);//////////////////////////////////
 fourbackrow2Dcpar2(psi1k, tempu, in, out, pback, m, n, communicator, numprocs, rank,
      MPI_complex);
 chebbackcol2Dcpar2(tempu, u,incheb,outcheb,pchebback, m, n, communicator, numprocs,
     rank, MPI_complex);

// for(i = 0;i<(m+1)*(n+1);i++)
// {u[i][1] = 0.0;}

 fftw_free(tempu);
 fftw_free(psi1k);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Assembles the v velocity field for cheb coeffs arranged column wise'p
int calcv(fftw_complex *psik, fftw_complex *v, fftw_complex *in, fftw_complex *out,
    fftw_plan pback,fftw_complex *incheb,fftw_complex *outcheb,fftw_plan pchebback,
    double Lfour,int m, int n, MPI_Datatype MPI_complex)
 {int k,colno,rowno,i;

 fftw_complex *tempv, *psi1k;
 tempv = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*(m+1));
 psi1k = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*(m+1));

 getfouru1krow2Dc (psik, psi1k,Lfour, m, n);
 fourbackrow2Dc (psi1k, tempv, in, out, pback, m, n, MPI_complex);
 chebbackcol2Dc(tempv, v,incheb,outcheb,pchebback, m, n, MPI_complex);

  for(i = 0;i<(m+1)*(n+1);i++)
        {v[i][0] = (-1.0)*v[i][0];}

 fftw_free(tempv);
 fftw_free(psi1k);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Assembles the v velocity field for cheb coeffs arranged column wise'p
int calcvpar(fftw_complex *psik, fftw_complex *v, fftw_complex *in, fftw_complex *out
    , fftw_plan pback,fftw_complex *incheb,fftw_complex *outcheb,fftw_plan pchebback,
     double Lfour,int m, int n, MPI_Datatype MPI_complex)
 {int k,colno,rowno,i;

 fftw_complex *tempv, *psi1k;
 tempv = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*(m+1));
 psi1k = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*(m+1));
```

```
 getfouru1krow2Dc (psik, psi1k,Lfour, m, n);
 fourbackrow2Dcpar (psi1k, tempv, in, out, pback, m, n, MPI_complex);
 chebbackcol2Dcpar(tempv, v,incheb,outcheb,pchebback, m, n, MPI_complex);

  for(i = 0;i<(m+1)*(n+1);i++)
        {v[i][0] = (-1.0)*v[i][0];}

 fftw_free(tempv);
 fftw_free(psi1k);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Assembles the v velocity field for cheb coeffs arranged column wise'p
int calcvpar2(fftw_complex *psik, fftw_complex *v, fftw_complex *in, fftw_complex *
    out, fftw_plan pback,fftw_complex *incheb,fftw_complex *outcheb,fftw_plan
    pchebback, double Lfour,int m, int n, MPI_Comm communicator, int numprocs, int
    rank, MPI_Datatype MPI_complex)
 {int k,colno,rowno,i;

 fftw_complex *tempv, *psi1k;
 tempv = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*(m+1));
 psi1k = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(n+1)*(m+1));

 getfouru1krow2Dc (psik, psi1k,Lfour, m, n);
 fourbackrow2Dcpar2(psi1k, tempv, in, out, pback, m, n, communicator, numprocs, rank,
      MPI_complex);
 chebbackcol2Dcpar2(tempv, v,incheb,outcheb,pchebback, m, n, communicator, numprocs,
     rank, MPI_complex);

  for(i = 0;i<(m+1)*(n+1);i++)
        {v[i][0] = (-1.0)*v[i][0];}

 fftw_free(tempv);
 fftw_free(psi1k);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Calculates and prints onto screen, the values of u and v
int printuv(fftw_complex *psik, fftw_complex *in, fftw_complex *out, fftw_plan pback,
    fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebback, double Lcheb,
    double Lfour, int m, int n, MPI_Datatype MPI_complex)
{fftw_complex *u, *v;

 u = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 v = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 calcu(psik, u, in, out, pback, incheb, outcheb, pchebback, Lcheb, m, n, MPI_complex)
     ;
 calcv(psik, v, in, out, pback, incheb, outcheb, pchebback, Lfour, m, n, MPI_complex)
     ;

 printf("\nu velocities: \n");print2Dc(u, (m+1),(n+1));
 printf("\nv velocities: \n");print2Dc(v, (m+1),(n+1));

 fftw_free(u);
 fftw_free(v);

 return 1;
 }
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Calculates and prints onto screen, the values of u and v
int fprintuv(FILE* fp,fftw_complex *psik, fftw_complex *in, fftw_complex *out,
    fftw_plan pback,fftw_complex *incheb, fftw_complex *outcheb, fftw_plan pchebback,
     double Lcheb, double Lfour, int m, int n, MPI_Datatype MPI_complex)
{fftw_complex *u, *v;

 u = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 v = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)*(m+1)*(n+1));
 calcu(psik, u, in, out, pback, incheb, outcheb, pchebback, Lcheb, m, n, MPI_complex)
     ;
 calcv(psik, v, in, out, pback, incheb, outcheb, pchebback, Lfour, m, n, MPI_complex)
     ;

 fprintf(fp,"\nu velocities: \n");fprint2Dc(fp,u, (m+1),(n+1));
 fprintf(fp,"\nv velocities: \n");fprint2Dc(fp,v, (m+1),(n+1));

 fftw_free(u);
 fftw_free(v);

 return 1;
 }
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Normalises the Boundary velocities
int normalisebound(double *boundaryval, double U, int n)
{double max = 0.0;
 int i;

// for(i = 0;i<2*(n+1);i++)
// if(max<fabs(boundaryval[i])) max = fabs(boundaryval[i]);

// (*U) = max;

 for(i = 0;i<2*(n+1);i++)
 boundaryval[i] = boundaryval[i]/(U);
// print2D(boundaryval,2,n+1);
return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//Checks if the solution has converged
//Returns 1 if convergence criteria has been reached
 int checkconvc(fftw_complex *u1, fftw_complex*u2, double error, int size)
{int i;
 double err0 = 0.0;
 double err1 = 0.0;
 for(i = 0;i<size;i++)
        {err0 += (u1[i][0] - u2[i][0])*(u1[i][0] - u2[i][0])/size;
         err1 += (u1[i][1] - u2[i][1])*(u1[i][1] - u2[i][1])/size;
        }
 if(sqrt(err0) < error && sqrt(err1) < error)
        return 1;
 else
        return 0;
}

int maxc(fftw_complex*u,int size)
{int i;
 double maxr,maxim;
 maxr = maxim =0.0;
```

```
 for(i = 0;i<size;i++)
        {if(fabs(u[i][0])>maxr) maxr = fabs(u[i][0]);
        if(fabs(u[i][1])>maxim) maxim = fabs(u[i][1]);
        }
 printf("\nMax Real = %g \t\t Max Imaginary = %g\n",maxr,maxim);
return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Returns the physical value from the coeffs when the cheb approx is columnwise and th fourier approximation is rowwise
double retphys(fftw_complex *uk,double x,double y,double Lfour,int m, int n)
{double u,Tl;
 double Tln,Tlnmin1;
 int rowno,colno,multiplier;
 u = 0.0;
 for(rowno = 0;rowno<2;rowno++)
        {//Tl = cos(rowno*acos(y));
        if(rowno == 0) Tl = 1.0;
        else if(rowno == 1) Tl = y;
        for(colno = 0;colno<n+1;colno++)
                {if(colno<ceil1(n+1,2)) multiplier = colno;else multiplier = (colno-n-1);
                u += Tl*(uk[rowno*(n+1)+colno][0]*cos(multiplier*x*2*PI/Lfour) - uk[rowno*(n+1)+colno][1]*sin(multiplier*x*2*PI/Lfour));
                }
        }

Tlnmin1 = 1.0;
Tln = Tl;

 for(rowno= 2;rowno<m+1;rowno++)
        {//Tl = cos(rowno*acos(y));
        Tl = 2.0*y*Tln - Tlnmin1;
//      for(colno = 0;colno<n+1;colno++)
        for(colno = 0;colno<ceil1(n+1,2);colno++)
                {//if(colno<ceil1(n+1,2)) multiplier = colno;else multiplier = (colno-n-1);
                multiplier = colno;
                u += Tl*(uk[rowno*(n+1)+colno][0]*cos(multiplier*x*2*PI/Lfour) - uk[rowno*(n+1)+colno][1]*sin(multiplier*x*2*PI/Lfour));
                }
        for(colno = ceil1(n+1,2);colno<n+1;colno++)
                {//if(colno<ceil1(n+1,2)) multiplier = colno;else multiplier = (colno-n-1);
                multiplier = (colno-n-1);
                u += Tl*(uk[rowno*(n+1)+colno][0]*cos(multiplier*x*2*PI/Lfour) - uk[rowno*(n+1)+colno][1]*sin(multiplier*x*2*PI/Lfour));
                }

        Tlnmin1 = Tln;
        Tln = Tl;
        }
 return u;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Returns the physical value from the coeffs when the cheb approx is columnwise and th fourier approximation is rowwise
int retphys1(fftw_complex *uk,double x,double y,double Lfour,double *u,int m, int n)
{double Tl;
 int rowno,colno,multiplier;
```

```
//print2Dc(uk,m+1,n+1);
 (*u) = 0.0;
 for(rowno= 0;rowno<m+1;rowno++)
        {Tl = cos(rowno*acos(y));
        for(colno = 0;colno<n+1;colno++)
                {if(colno<ceil1(n+1,2)) multiplier = colno;else multiplier = (colno-n
                    -1);
                (*u) += Tl*(uk[rowno*(n+1)+colno][0]*cos(multiplier*x*2*PI/Lfour) -
                    uk[rowno*(n+1)+colno][1]*sin(multiplier*x*2*PI/Lfour));
                }
        }
//printf("\n u = %g \tx = %g \ty = %g \tLfour = %g \tm = %d \t n = %d \tTl = %g\n",(*
    u),x,y,Lfour,m,n,Tl);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Returns the array of physical values
int retphysmat(double *u, double *coord, int coordn, fftw_complex *uk,double Lfour,
    int m, int n)
{int i;
 for(i = 0;i<coordn;i++)
        u[i] = retphys(uk,coord[2*i],coord[2*i+1],Lfour,m,n);
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
double retu(fftw_complex *uk, double Uref,double href, double x, double y,double
    Lfour, int m, int n)
{double xstar,ystar,Ustar,u;
 xstar = x/href;
 ystar = y/href;

retphys1(uk,xstar,ystar,Lfour,&Ustar,m,n);
 *(&u) = Ustar*Uref;
//printf("\nu = %g  xstar = %g  ystar = %g  Ustar = %g m =%d  n = %d \tLfour = %g\n",
    u,xstar,ystar,Ustar,m,n,Lfour);

 return u;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int retu1(fftw_complex *uk, double Uref,double href, double x, double y,double Lfour,
     double *u,int m, int n)
{double xstar,ystar,Ustar;
 xstar = x/href;
 ystar = y/href;

retphys1(uk,xstar,ystar,Lfour,&Ustar,m,n);
 (*u) = Ustar*Uref;
//printf("\nu = %g  xstar = %g  ystar = %g  Ustar = %g m =%d  n = %d \tLfour = %g\n",
    u,xstar,ystar,Ustar,m,n,Lfour);

 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
double retpsi(fftw_complex *psik, double Uref,double href, double x, double y,double
    Lfour, int m, int n)
```

```
{double xstar,ystar,psistar,psi;
 xstar = x/href;
 ystar = y/href;

psistar = retphys(psik,xstar,ystar,Lfour,m,n);
 psi = psistar*Uref*href;
 return psi;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int retpsi1(fftw_complex *psik, double Uref,double href, double x, double y,double
    Lfour,double *psi, int m, int n)
{double xstar,ystar,psistar;
 xstar = x/href;
 ystar = y/href;

psistar = retphys(psik,xstar,ystar,Lfour,m,n);
 (*psi) = psistar*Uref*href;
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int getn1n2(int *n1, int *n2, int rank,int *tasks, int *tasksnum, int *numtasks,int
    numprocs, int N)
{int rem,i,j,k;
 rem = N- (N/numprocs)*numprocs;

 if(numprocs >N)
        (*numtasks) = N;
 else
        (*numtasks) = numprocs;
 if(numprocs>N || numprocs ==N)
        {(*n1)=(*n2)=rank;
//      if(rank == 0)
                for(i = 0;i<numprocs;i++)
                        {if(i<(*numtasks))
                                {tasks[i] = i;
                                tasksnum[i]=1;
                                }
                        else
                                {tasks[i] = -1;
                                tasksnum[i] = 0;
                                }
                        }
        }
else if(rank> numprocs - rem)
                {(*n1) = (N/numprocs) * (numprocs - rem) + (rank - (numprocs - rem))
                    *(N/numprocs+1);
                 (*n2) = (*n1)+(N/numprocs)+1-1;
                }
        else if (rank == numprocs - rem)
                        {(*n1) = (N/numprocs)*rank;
                        (*n2) = (*n1) + (N/numprocs)+1-1;
                        }
                else
                        {(*n1) = (N/numprocs)*rank;
                        (*n2) = (*n1) + (N/numprocs)-1;
                        }
//if (rank == 0)
        if(N > numprocs)
        for(i = 0;i<numprocs;i++)
                {if(i> numprocs - rem)
```

```
                        {tasks[i] = (N/numprocs) * (numprocs - rem) + (i - (numprocs
                            - rem))*(N/numprocs+1);
                        tasksnum[i] = (N/numprocs)+1;
                        }
                else if (i == numprocs - rem)
                                {tasks[i] = (N/numprocs)*i;
                                tasksnum[i] = (N/numprocs)+1;
                                }
                        else
                                {tasks[i] = (N/numprocs)*i;
                                tasksnum[i] = (N/numprocs);
                                }
                }

/*printf("\nFor process# %d: (*n1) = %d, (*n2) = %d", rank,(*n1),(*n2));
if(rank == 0)
        {printf("\n Tasks: \t");print2Dint(tasks,1,numprocs);
        printf("\n Tasksnum: \t");print2Dint(tasksnum,1,numprocs);
        }
*/
//printf("\nrem = %d, n1 = %d, n2 = %d",rem,(*n1),(*n2));
//printf("\ntasks = \t");print2Dint(tasks,1,numtasks);printf("\ntasksnum = \t");
    print2Dint(tasksnum,1,numtasks);
return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int retseglengths(int *tasks, int *tasksnum, int nelements, int ndiv)
{int i, rem;
  rem = nelements - (nelements/ndiv)*ndiv;
        for(i = 0;i<ndiv;i++)
                {if(i> ndiv - rem)
                        {tasks[i] = (nelements/ndiv) * (ndiv - rem) + (i - (ndiv -
                            rem))*(nelements/ndiv+1);
                        tasksnum[i] = (nelements/ndiv)+1;
                        }
                else if (i == ndiv - rem)
                                {tasks[i] = (nelements/ndiv)*i;
                                tasksnum[i] = (nelements/ndiv)+1;
                                }
                        else
                                {tasks[i] = (nelements/ndiv)*i;
                                tasksnum[i] = (nelements/ndiv);
                                }
                }
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int changeres(fftw_complex *source, fftw_complex *dest, int ys, int xs, int yd, int
    xd)
{int colno, rowno,i,j;
 int ncol,nrow, one = 1,nk,nkmin1,min1 = -1,two = 2; double alpha = -1.0;

 if(ys > yd) nrow = yd; else nrow = ys;
 if(xs > xd) ncol = xd; else ncol = xs;

 nk = ceil1(ncol,2);
 nkmin1 = nk-1;
 makezeroc(dest,xd*yd);
/* for(colno=0;colno<ceil1(xs,2);colno++)
        for(rowno = 0; rowno<ys; rowno++)
                if(colno<ceil1(xd,2) && rowno<yd)
                {dest[rowno*xd+colno][0] = source[rowno*xs+colno][0];
                dest[rowno*xd+colno][1] = source[rowno*xs+colno][1];
                }
```

```
*/
for(rowno = 0; rowno<nrow; rowno++)
        zcopy_(&nk, &(source[rowno*xs]),&one, &(dest[rowno*xd]),&one);

/*      for(colno = 1; colno<ceil1(xd,2);colno++)
                for(j =0;j<yd;j++)
                        {dest[j*(xd)+(xd)-colno][0] = dest[j*(xd)+colno][0];
                         dest[j*(xd)+(xd)-colno][1] = -dest[j*(xd)+colno][1];
                        }
*/
for(rowno = 0; rowno<nrow; rowno++)
        {zcopy_(&nkmin1, &(dest[rowno*xd+1]),&one, &(dest[rowno*xd+xd-nkmin1]),&min1)
            ;
        dscal_(&nkmin1, &alpha, &(dest[rowno*xd+xd-nkmin1][1]), &two);
        }

/* for(colno=ceil1(xs,2);colno<xs;colno++)
        for(rowno = 0; rowno<ys; rowno++)
                {dest[rowno*xd + ceil1(xd,2)+colno-ceil1(xs,2)][0] = source[rowno*xs+
                    colno][0];
                dest[rowno*xd + ceil1(xd,2)+colno-ceil1(xs,2)][1] = source[rowno*xs+
                    colno][1];
                }
*/
 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
int changeres2(double *source, double *sourcei, fftw_complex *dest, int ys, int xs,
    int yd, int xd)
{int colno, rowno,i;
 int ncol,nrow, one = 1,nk,nkmin1,min1 = -1,two = 2; double alpha = -1.0;

 if(ys > yd) nrow = yd; else nrow = ys;
 if(xs > xd) ncol = xd; else ncol = xs;

 nk = ceil1(ncol,2);
 nkmin1 = nk-1;

 makezeroc(dest,xd*yd);
/* for(colno=0;colno<ceil1(xs,2);colno++)
        for(rowno = 0; rowno<ys; rowno++)
                {dest[rowno*xd+colno][0] = source[rowno*xs+colno];
                dest[rowno*xd+colno][1] = sourcei[rowno*xs+colno];
                }
*/
for(rowno = 0; rowno<nrow; rowno++)
        dcopy_(&nk, &(source[rowno*xs]),&one, &(dest[rowno*xd][0]),&two);
for(rowno = 0; rowno<nrow; rowno++)
        dcopy_(&nk, &(sourcei[rowno*xs]),&one, &(dest[rowno*xd][1]),&two);
/* for(colno=ceil1(xs,2);colno<xs;colno++)
        for(rowno = 0; rowno<ys; rowno++)
                {dest[rowno*xd + ceil1(xd,2)+colno-ceil1(xs,2)][0] = source[rowno*xs+
                    colno];
                dest[rowno*xd + ceil1(xd,2)+colno-ceil1(xs,2)][1] = sourcei[rowno*xs+
                    colno];
                }
*/
for(rowno = 0; rowno<nrow; rowno++)
        {zcopy_(&nkmin1, &(dest[rowno*xd+1]),&one, &(dest[rowno*xd+xd-nkmin1]),&min1)
            ;
        dscal_(&nkmin1, &alpha, &(dest[rowno*xd+xd-nkmin1][1]), &two);
```

```
        }

 return 1;
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Returns the array of physical values
int retphysmatpar(double *u, double *coord, unsigned long int coordn, fftw_complex *uk,double Lfour, int rank, int numprocs, int m, int n, MPI_Datatype MPI_complex)
{int i;
 int n1,n2,*tasks, *tasksnum,numtasks, N;
 double *utemp;

//  int blockcounts[1];
  MPI_Status Stat;
/*  MPI_Datatype MPI_complex,oldtypes[1];
  MPI_Aint offsets[1];
 offsets[0] = 0;
 blockcounts[0] = 2;
 oldtypes[0] = MPI_DOUBLE;
 MPI_Type_struct(1,blockcounts,offsets,oldtypes,&MPI_complex);
 MPI_Type_commit (&MPI_complex);
*/

 tasks = malloc(sizeof(int)*numprocs);
 tasksnum = malloc(sizeof(int)*numprocs);

// N = coordn;

 getn1n2(&n1, &n2, rank, tasks, tasksnum, &numtasks, numprocs, coordn);
 utemp = malloc(sizeof(double)*(n2-n1+1));

// printf("\nrank = %d, n1 = %d, n2 = %d, numtasks = %d\n",rank,n1,n2,numtasks);
// if(rank == 0){printf("\ntasks = \t");print2Dint(tasks,1,numtasks);printf("\ntasksnum = \t");print2Dint(tasksnum,1,numtasks);}

////////////  MPI_Bcast(uk,(m+1)*(n+1),MPI_complex,0,MPI_COMM_WORLD);
// MPI_Barrier(MPI_COMM_WORLD);
//printf("\nReached: #%d\n",rank);


 for(i = 0;i<n2-n1+1;i++)
        utemp[i] = retphys(uk,coord[2*(i+n1)],coord[2*(i+n1)+1],Lfour,m,n);

 if(rank<numtasks && rank !=0)
 MPI_Send(utemp,n2-n1+1,MPI_DOUBLE, 0,rank,MPI_COMM_WORLD);
//printf("\n#%d Reached\n",rank);
// MPI_Barrier(MPI_COMM_WORLD);
 if(rank ==0)
        {for(i = 1;i<numtasks;i++)
                MPI_Recv(&(u[tasks[i]]),tasksnum[i],MPI_DOUBLE,i,i,MPI_COMM_WORLD,&Stat);
        for(i = 0; i <(n2-n1+1);i++)
                {u[i] = utemp[i]; }
        }

// MPI_Barrier(MPI_COMM_WORLD);
 MPI_Bcast(u,coordn,MPI_DOUBLE,0,MPI_COMM_WORLD);
 free(tasks);
 free(tasksnum);
 free(utemp);
 return 1;
```

```
}                                                                                    2033
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++2034
                                                                                     2035
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++2036
int copyarray(double *source, double *dest, int size)                                2037
{int i;                                                                              2038
 int one = 1;                                                                        2039
// for(i = 0;i<size;i++)                                                             2040
//       dest[i] = source[i];                                                        2041
 dcopy_(&size, source, &one, dest, &one);                                            2042
 return 1;                                                                           2043
}                                                                                    2044
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++2045
```

VITA

Pradeep Chandrakant Rao was born in India in 1983. He graduated with a B.E. degree from Mumbai University in 2005. Upon graduation, he worked for 8 months with Godrej and Boyce Manufacturing Ltd, in the Compressed Air Solutions Department. Subsequently he worked for one year and 8 months at Uhde India Ltd in the capacity of Piping Engineer. He began his master's program in the Mechanical Engineering Department at Texas A&M University in Jan of 2008, and joined the Fluids, Turbulence and Fundamental Transport Lab (FT2L) under Dr. Andrew Duggleby in May of 2008. He graduated in December 2009.

He may be reached at:

Mechanical Engineering Dept.
c/o Dr. Andrew Duggleby
Texas A&M University
College Station, TX 77843-3123.

The typist for this thesis was the author.