An efficient spectral method for simulation of incompressible flow over a flat plate

By **Anders Lundbladh**[*†], **Stellan Berlin**[†], **Martin Skote**[†], **Casper Hildings**[*†], **Jaisig Choi**[‡], **John Kim**[‡] and **Dan S. Henningson**[*†]

An efficient spectral integration technique for the solution of the Navier-Stokes equations for incompressible flow over a flat plate is described and implemented in a computer code using the FORTRAN language. The algorithm can either be used for temporal or spatial simulation. In the latter case, a fringe region technique is used to allow a streamwise inflow and outflow of the computational domain. At a constant distance from the flat plate an artificial boundary is introduced and a free-stream boundary condition applied. The plate parallel directions are discretized using Fourier series and the normal direction using Chebyshev series. Time integration is performed using third order Runge-Kutta method for the advective and forcing terms and Crank-Nicolson for the viscous terms. The version of the code described in this report can be run on parallel computers with shared memory. A slightly different version also exists which utilizes MPI (Message-Passing Interface) for parallelization on distributed memory computers.

[*]Aeronautical Research Institute of Sweden,Box 11021, SE-161 11 Bromma, Sweden
[†]Department of Mechanics, KTH, SE-100 44 Stockholm, Sweden
[‡]Department of Mechanical and Aero Space, UCLA, 405 Hilgard Ave, LA, CA 90095, USA

Contents

1. Introduction

Solution of the Navier-Stokes equations for the simulation of transition and turbulence requires high numerical accuracy for a large span of length scales. This has prompted a development of accurate spectral methods. Unfortunately even with these methods computations require an immense amount of computer time and memory. In the present report we use spectral methods to derive an accurate algorithm of the flat plate boundary layer flow geometry. The basic numerical method is similar to the Fourier-Chebyshev method used by Kim *et al.* (1987).

The original algorithm (Lundbladh *et al.* 1992*a*) solved the incompressible flow equations in a channel flow geometry. To allow simulations of the flow over a flat plate a free-stream boundary condition is required, and for spatial simulations a fringe region technique similar to that of Bertolotti *et al.* (1992) is described.

For further details about spectral discretizations and additional references see Canuto *et al.* (1988).

The original channel code and the implementation of the present numerical method has been used in a number of investigations.
In channel flow:
Henningson *et al.* (1990), Lu & Henningson (1990), Lundbladh & Johansson (1991), Schmid & Henningson (1992), Lundbladh (1993), Henningson *et al.* (1993), Lundbladh & Henningson (1993), Schmid & Henningson (1993), Elofsson & Lundbladh (1994), Kreiss *et al.* (1994), Lundbladh *et al.* (1994*a*), Schmid *et al.* (1994), Henningson (1995), Reddy *et al.* (1998).
In boundary layer flow:
Lundbladh *et al.* (1992*b*), Berlin *et al.* (1994), Henningson & Lundbladh (1994), Lundbladh *et al.* (1994*b*), Henningson & Lundbladh (1995), Högberg & Henningson (1998), Schmid *et al.* (1996), Nordström *et al.* (1999), Hildings (1997), Berlin & Henningson (1999), Berlin *et al.* (1998*a*), Berlin *et al.* (1999), Berlin *et al.* (1998*b*), Bech *et al.* (1998), Skote *et al.* (1998).

2. The numerical method

2.1. *Derivation of the velocity-vorticity formulation*

The starting point is the non-dimensionalized incompressible Navier-Stokes equations in a rotating reference frame, here written in tensor notation,

$$\frac{\partial u_i}{\partial t} = -\frac{\partial p}{\partial x_i} + \epsilon_{ijk} u_j (\omega_k + 2\Omega_k) - \frac{\partial}{\partial x_i}(\frac{1}{2} u_j u_j) + \frac{1}{R}\nabla^2 u_i + F_i, \qquad (1)$$

$$\frac{\partial u_i}{\partial x_i} = 0, \qquad (2)$$

with boundary conditions at the flat plate and at the free-stream boundary, which are discussed in the next subsections.

The first equation represents conservation of momentum and the second equation incompressibility of the fluid. Here $(x_1, x_2, x_3) = (x, y, z)$ are the streamwise, normal and spanwise coordinates, $(u_1, u_2, u_3) = (u, v, w)$ are the respective velocities, $(\omega_1, \omega_2, \omega_3) = (\chi, \omega, \vartheta)$ are the corresponding vorticities, and p is the pressure. The streamwise and spanwise directions will alternatively be termed horizontal directions. Ω_k is the angular velocity of the coordinate frame around axis k. In practise the most often used case is rotation around the spanwise axis, thus let $\Omega = \Omega_3$ be the rotation number. F_i is a body force which is used for numerical purposes that will be further discussed below. It can also be used to introduce disturbances in the flow. The Reynolds number is defined as $R = U_\infty \delta^* / \nu$, where U_∞ is the undisturbed streamwise free-stream velocity at $x = 0$ and $t = 0$, δ^* is the displacement thickness of the undisturbed streamwise velocity at $x = 0$ and $t = 0$, and ν is the kinematic viscosity. The size of the solution domain in physical space is x_L, y_L and z_L in the streamwise, normal and spanwise directions, respectively.

A Poisson equation for the pressure can be obtained by taking the divergence of the momentum equation,

$$\nabla^2 p = \frac{\partial H_i}{\partial x_i} - \nabla^2(\frac{1}{2} u_j u_j), \qquad (3)$$

where $H_i = \epsilon_{ijk} u_j (\omega_k + 2\Omega_k) + F_i$. Application of the Laplace operator to the momentum equation for the normal velocity yields an equation for that component through the use of equations (3) and (2). One finds

$$\frac{\partial \nabla^2 v}{\partial t} = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial z^2}\right) H_2 - \frac{\partial}{\partial y}(\frac{\partial H_1}{\partial x} + \frac{\partial H_3}{\partial z}) + \frac{1}{R}\nabla^4 v. \qquad (4)$$

This equation can, for numerical purposes, be written as a system of two second order equations:

$$\begin{aligned} \frac{\partial \phi}{\partial t} &= h_v + \frac{1}{R}\nabla^2 \phi, \\ \nabla^2 v &= \phi, \end{aligned} \tag{5}$$

where

$$h_v = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial z^2}\right) H_2 - \frac{\partial}{\partial y}\left(\frac{\partial H_1}{\partial x} + \frac{\partial H_3}{\partial z}\right). \tag{6}$$

An equation for the normal vorticity can be found by taking the curl of the momentum equation. The second component of that equation reads,

$$\frac{\partial \omega}{\partial t} = h_\omega + \frac{1}{R}\nabla^2 \omega, \tag{7}$$

where

$$h_\omega = \frac{\partial H_1}{\partial z} - \frac{\partial H_3}{\partial x}. \tag{8}$$

Note that the equations for ϕ, v and ω have similar form, and can thus be solved using the same numerical routine. Once the the normal velocity v and the normal vorticity ω have been calculated the other velocity components can be found form the incompressibility constraint and the definition of the normal vorticity.

2.2. *Boundary condition*

The boundary conditions in the horizontal directions are periodic but we need to specify boundary conditions at the plate and in the free-stream, to solve equations (5) and (7). The natural no-slip boundary conditions read

$$v(y=0) = 0, \ \frac{\partial v(y=0)}{\partial y} = 0, \ \omega(y=0) = 0. \tag{9}$$

For disturbance generation and control by blowing and suction through the plate, an arbitrary time dependent velocity distribution,

$$v(y=0) = v_{BS}(x, z, t), \tag{10}$$

can be used.

The flow is assumed to extend to an infinite distance perpendicularly to the flat plate. However, the discretization discussed below can only handle a finite domain. Therefore, the flow domain is truncated and an artificial boundary condition is applied in the free-stream.

The simplest possible is a Dirichlet condition i.e.,

$$u_i(y = y_L) = \mathcal{U}_i(y = y_L), \tag{11}$$

where $\mathcal{U}_i(x,y)$ is a base flow that is normally chosen as a Falkner-Skan-Cook flow. An arbitrary pressure gradient, to for instance create a separation bubble, can be imposed by choosing $\mathcal{U}_i$ accordingly.

The desired flow solution generally contains a disturbance and that will be forced to zero by the Dirichlet condition. This introduces an error compared to the exact solution for which the boundary condition is applied at an infinite distance from the wall. The error may result in increased damping for disturbances in the boundary layer.

Some improvement can be achieved by using a Neumann condition,

$$\frac{\partial u_i}{\partial y}|_{y=y_L} = \frac{\partial \mathcal{U}_i}{\partial y}|_{y=y_L}. \tag{12}$$

This condition can be shown to be stable if there is outflow at the boundary or the inflow is weaker than $O(1/R)$. This restriction is usually fulfilled if the boundary is placed on a sufficiently large distance from the wall, so that the disturbance velocity is small.

A generalization of the boundary condition used by Malik *et al.* (1985) allows the boundary to be placed closer to the wall. It is an asymptotic condition that decreases the error further and it reads,

$$\left[\frac{\partial \hat{u}_i}{\partial y} + |k|\hat{u}_i\right]_{y=y_L} = \left[\frac{\partial \hat{\mathcal{U}}_i}{\partial y} + |k|\hat{\mathcal{U}}_i\right]_{y=y_L}, \tag{13}$$

where $\hat{}$ denotes the horizontal Fourier transform with respect to the horizontal coordinates, $k^2 = \alpha^2 + \beta^2$ and α and β are the horizontal wavenumbers (see equation 29). Thus this condition is most easily applied in Fourier space. The boundary condition exactly represents a potential flow solution decaying away from the wall. It is essentially equivalent to requiring that the vorticity is zero at the boundary. Thus, it can be applied immediately outside the vortical part of the flow.

2.3. *Forcing for temporal simulation*

A localized disturbance or wave of relatively short wavelength which travels downstream in a slowly growing boundary layer is surrounded by a boundary layer of almost constant thickness which grows slowly in time. This forms the basis of the temporal simulation technique.

Following the ideas of Spalart & Yang (1987) we assume that the boundary layer streamwise velocity is $U(x,y)$ and introduce a reference point $x_r = x_0 + ct$ where c is a reference speed. We now assume that the undisturbed boundary layer in the vicinity of the disturbance has the velocity distribution $U(y,t) = U(x_r,y)$, $V(y,t) = 0$. Since the boundary layer is now parallel (as there is no dependence on x), it is possible to apply periodic boundary conditions in the horizontal directions. However, whereas $U(x,y)$ (with the corresponding

V given by continuity) is a solution to Navier-Stokes or at least the boundary layer equations, this is not true for $\{U(y,t), V(y,t)\}$. Thus to ensure the correct development of the boundary layer profile over extended periods of time it is necessary to add a (weak) forcing to balance the streamwise momentum equation,

$$F_1 = \frac{\partial U(y,t)}{\partial t} - \frac{1}{R}\frac{\partial^2 U(y,t)}{\partial y^2} = c\frac{\partial U(x,y)}{\partial x} - \frac{1}{R}\frac{\partial^2 U(x,y)}{\partial y^2}, \tag{14}$$

where the right hand side should be evaluated at the reference coordinate x_r. The reference speed should be chosen as the group speed of the wave or the propagation speed of the localized disturbance for best agreement with a spatially developing flow. To fully justify the periodic boundary conditions in the case of a wave train, the wave itself should be slowly developing.

2.4. *Forcing for spatial simulation*

The best numerical model of a physical boundary layer, which is usually developing in the downstream direction rather than in time, is a spatial formulation. To retain periodic boundary conditions, which is necessary for the Fourier discretization described below, a fringe region is added downstream of the physical domain, similar to that described by Bertolotti *et al.* (1992). In the fringe region disturbances are damped and the flow returned to the desired inflow condition. This is accomplished by the addition of a volume force which only increases the execution time of the algorithm by a few percent.

The form of the forcing is :

$$F_i = \lambda(x)(\mathcal{U}_i - u_i), \tag{15}$$

where $\lambda(x)$ is a non-negative fringe function which is significantly non-zero only within the fringe region. $\mathcal{U}_i$ is the same flow field used for the boundary conditions, which also contains the desired flow solution in the fringe. The streamwise velocity component is calculated as,

$$\mathcal{U}_x(x,y) = U(x,y) + [U(x+x_L,y) - U(x,y)]\, S\left(\frac{x - x_{mix}}{\Delta_{mix}}\right), \tag{16}$$

where $U(x,y)$ is normally a solution to the boundary layer equations. Here x_{mix} and Δ_{mix} are chosen so that the prescribed flow, within the fringe region, smoothly changes from the outflow velocity of the physical domain to the desired inflow velocity. S is given below. The wall normal component $\mathcal{U}_y$ is then calculated from the equation of continuity, and the spanwise velocity $\mathcal{U}_z$ is set to zero for simulations where the mean flow is two dimensional. For three dimensional boundary layers $\mathcal{U}_z$ is computed from a boundary layer solution in fashion analogous to that for $\mathcal{U}_x$. This choice of $\mathcal{U}$ ensures that for the undisturbed laminar boundary layer the decrease in thickness is completely confined

to the fringe region, thus minimizing the upstream influence. A forced disturbance to the laminar flow can be given as inflow condition if that disturbance is included in $\mathcal{U}_i$.

A convenient form of the fringe function λ is as follows,

$$\lambda(x) = \lambda_{max}[S\left(\frac{x - x_{start}}{\Delta_{rise}}\right) - S\left(\frac{x - x_{end}}{\Delta_{fall}} + 1\right)]. \tag{17}$$

Here λ_{max} is the maximum strength of the damping, x_{start} to x_{end} the spatial extent of the region where the damping function is non-zero and Δ_{rise} and Δ_{fall} the rise and fall distance of the damping function. $S(x)$ is a smooth step function rising from zero for negative x to one for $x \geq 1$. We have used the following form for S, which has the advantage of having continuous derivatives of all orders.

$$S(x) \quad = \quad \begin{cases} 0 & x \leq 0 \\ 1/[1 + \exp(\frac{1}{x-1} + \frac{1}{x})] & 0 < x < 1 \\ 1 & x \geq 1 \end{cases} . \tag{18}$$

To achieve maximum damping both the total length of the fringe and λ_{max} have to be tuned. The actual shape of $\lambda(x)$ is less important for the damping but it should have its maximum closer to x_{end} than to x_{start}. The damping is also strongly effected by the resolution of the disturbance that should be damped. An investigation of how the fringe parameters effect the disturbance in the fringe can be found in Appendix C.

For maximum computational efficiency the simulated flow has to be considered when the fringe parameters are tuned. Assuming that the achieved damping is sufficient, a short fringe reduces the box length and therefore the required CPU time per iteration. However, if the flow gradients introduced in the fringe region are larger than those in the physical domain that may decrease the time step and consequently increase the necessary number of iterations. Note that the boundary layer growth causes outflow through the free-stream boundary. The streamwise periodicity requires that all that fluid enters in the fringe region.

Analysis of Navier-Stokes equations with a fringe forcing term yields that there is an additional part of the disturbance associated with the pressure whose decay is not dependent on λ. For a boundary layer, this solution decays appreciably over a downstream distance equal to the boundary layer thickness, and thus the fringe region must be some factor (say 10 to 30) times this thickness to get a large decay factor, see Nordström *et al.* (1999).

2.5. *Temporal discretization*

The time advancement is carried out by one of two semi-implicit schemes. We illustrate them on the equation

$$\frac{\partial \psi}{\partial t} = G + L\psi, \tag{19}$$

	$a_n/\Delta t^n$	$b_n/\Delta t^n$	$c_n/\Delta t^n$
RK3	8/15	0	0
3-stage	5/12	-17/60	8/15
	3/4	-5/12	2/3
RK3	8/17	0	0
4-stage	17/60	-15/68	8/17
	5/12	-17/60	8/15
	3/4	-5/12	2/3

TABLE 1. Time stepping coefficients.

which is on the same form as equation (5) and (7). ψ represents ϕ or ω, G contains the (non-linear) advective, rotation and forcing terms and depends on all velocity and vorticity components, L is the (linear) diffusion operator. L is discretized implicitly using the second order accurate Crank-Nicolson (CN) scheme and G explicitly by a low storage third order three or four stage Runge-Kutta (RK3) scheme. These time discretizations may be written in the following manner : (G and L are assumed to have no explicit dependence on time)

$$\psi^{n+1} = \psi^n + a_n G^n + b_n G^{n-1} + (a_n + b_n)\left(\frac{L\psi^{n+1} + L\psi^n}{2}\right), \qquad (20)$$

where the constants a_n and b_n are chosen according to the explicit scheme used. The two possibilities for the RK3 schemes are shown in the table 1. Note that the RK3 schemes have three or four stages which imply that a full physical time step is only achieved every three or four iterations. The time used for the intermediate stages are given by $t = t + c_n$, where c_n is given in table 1.

To obtain some insight into the properties of these discretizations they will be applied to the two dimensional linearized Burgers' equation with a system rotation. The eigenvalue analysis yields a necessary condition for stability which must be augmented by an experimental verification. Putting the equation into the form of equation (19) yields :

$$\begin{aligned}
\psi &= \begin{bmatrix} u \\ w \end{bmatrix}, \\
G &= \begin{bmatrix} u_0\partial/\partial x + w_0\partial/\partial z & 2\Omega \\ -2\Omega & u_0\partial/\partial x + w_0\partial/\partial z \end{bmatrix} \begin{bmatrix} u \\ w \end{bmatrix}, \\
L &= \frac{1}{R}\begin{bmatrix} \partial^2/\partial x^2 + \partial^2/\partial z^2 & 0 \\ 0 & \partial^2/\partial x^2 + \partial^2/\partial z^2 \end{bmatrix}. \qquad (21)
\end{aligned}$$

It can be seen as an approximation to equation (1). The dependence of ψ on both the streamwise and spanwise coordinate directions have been included in

order to indicate how multiple dimensions enter into the stability considerations.

We will for simplicity use Fourier discretization in the spatial directions. The Chebyshev method acts locally as a transformed Fourier method and thus the stability properties derived here can be applied with the local space step. An exception to this occurs at the endpoints where the transformation is singular. It can be shown that the Chebyshev method is more stable there. A numerical study of a 1-dimensional advection equation using the Chebyshev discretization yields that the upper limit of its spectrum along the imaginary axis is about 16 times lower than the simple application of the results from the Fourier method. This allows a corresponding increase of the time step when the stability is limited by the wall normal velocity at the free-stream boundary.

Fourier transforming in x and z yields:

$$\hat{\psi}_t = \begin{bmatrix} i\alpha u_0 + i\beta w_0 & 2\Omega \\ -2\Omega & i\alpha u_0 + i\beta w_0 \end{bmatrix} \hat{\psi} - \frac{\alpha^2+\beta^2}{R}\hat{\psi}, \tag{22}$$

where α and β are the wavenumbers in the x- and z-directions, respectively. This equation can be diagonalized to yield the equation,

$$\hat{u}_t = i(\alpha u_0 + \beta w_0 \pm 2\Omega)\hat{u} + \frac{\alpha^2+\beta^2}{R}\hat{u}. \tag{23}$$

We assume that the absolute stability limit will first be reached for the largest wavenumbers of the discretization α_{max} and β_{max}, which corresponds to a wavelength of $2\cdot\Delta x$ and $2\cdot\Delta z$, respectively. Δx and Δz are the discretization step lengths in physical space. The following parameters are useful for our analysis,

$$\begin{aligned} \mu &= \Delta t[2|\Omega_k| + (\alpha_{max}|u_0| + \beta_{max}|w_0|)] \\ &= \Delta t\left[2|\Omega_k| + \pi\left(\frac{|u_0|}{\Delta x} + \frac{|w_0|}{\Delta z}\right)\right], \end{aligned} \tag{24}$$

$$\begin{aligned} \lambda &= \frac{1}{R}\Delta t(\alpha_{max}^2 + \beta_{max}^2) \\ &= \frac{1}{R}\pi^2\Delta t\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta z^2}\right). \end{aligned} \tag{25}$$

The parameter μ is usually called the spectral CFL number, in analogy with the stability theory for finite difference equations. Henceforth it will be termed simply the CFL number. Using the RK3-CN time discretization we have the

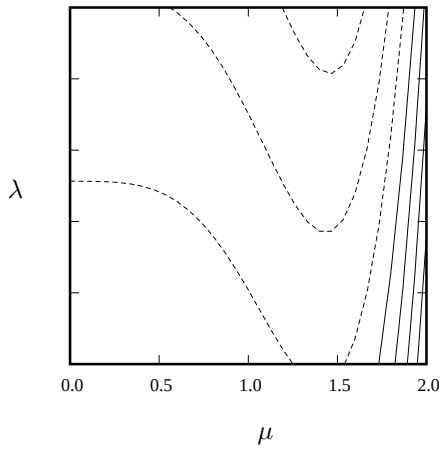

FIGURE 1. Contours of constant amplification factor for the RK3-CN method. Contour spacing is 0.05 with dashed lines indicating that the amplification factor is below unity.

following three stages in each time step for the model equation (23),

$$\begin{aligned}
\hat{u}^{n+1} &= \hat{u}^n + i\mu a_1 \hat{u}^n - \frac{\lambda}{2} a_1 (\hat{u}^{n+1} + \hat{u}^n), \\
\hat{u}^{n+2} &= \hat{u}^{n+1} + i\mu (a_2 \hat{u}^{n+1} + b_2 \hat{u}^n) - \frac{\lambda}{2}(a_2 + b_2)(\hat{u}^{n+2} + \hat{u}^{n+1}), \qquad (26) \\
\hat{u}^{n+3} &= \hat{u}^{n+2} + i\mu (a_3 \hat{u}^{n+2} + b_3 \hat{u}^{n+1}) - \frac{\lambda}{2}(a_3 + b_3)(\hat{u}^{n+3} + \hat{u}^{n+2}).
\end{aligned}$$

The absolute stability regions, i.e. the regions where all solutions to the above difference equations are bounded in the μ – λ plane, can now be found by calculating the roots of the associated characteristic polynomials. Contours of constant absolute values of the roots are given in figure 1. Figure 1 shows the curves for the RK3-CN method. Note that higher values of λ (lower Reynolds number) stabilizes the method, i.e. increases the CFL number (μ) that is allowed for an absolutely stable solution. In the limit of infinite Reynolds number the RK3-CN method approaches the limit $\sqrt{3}$, a result which also can be arrived at through the standard analysis of the RK3 scheme alone. The analysis for the four stage method is analogous and the stability limit is $\sqrt{8}$.

If the time advancement scheme (20) is applied to equations (5) and (7) we find (for the moment disregarding the boundary conditions),

$$\begin{aligned}
(1 - \frac{a_n + b_n}{2R}\nabla^2)\phi^{n+1} &= (1 + \frac{a_n + b_n}{2R}\nabla^2)\phi^n + a_n h_v^n + b_n h_v^{n-1}, \\
\nabla^2 v^{n+1} &= \phi^{n+1}, \qquad (27)
\end{aligned}$$

and

$$(1 - \frac{a_n + b_n}{2R}\nabla^2)\omega^{n+1} = (1 + \frac{a_n + b_n}{2R}\nabla^2)\omega^n + a_n h_\omega^n + b_n h_\omega^{n-1}. \quad (28)$$

2.6. *Horizontal discretization – Fourier expansions*

The discretization in the horizontal directions uses a Fourier series expansions which assumes that the solution is periodic.

The streamwise and spanwise dependence of each variable can then be written

$$u(x,z) = \sum_{l=-(\frac{N_x}{2}-1)}^{\frac{N_x}{2}-1} \sum_{m=-(\frac{N_z}{2}-1)}^{\frac{N_z}{2}-1} \hat{u}(\alpha,\beta)\exp[i(\alpha_l x + \beta_m z)], \quad (29)$$

where $\alpha_l = 2\pi l/x_L$ and $\beta_m = 2\pi m/z_L$, and N_x and N_z are the number of Fourier modes included in the respective directions. Note that the indices on the discrete wavenumbers α and β are sometimes left out for notational convenience and that $k^2 = \alpha^2 + \beta^2$.

2.6.1. *Normal velocity and normal vorticity equations*

Expanding the dependent variables of equation (27) in Fourier series gives

$$\begin{aligned}\left(1 - \frac{a_n + b_n}{2R}(D^2 - k^2)\right)\hat{\phi}^{n+1} &= \left(1 + \frac{a_n + b_n}{2R}(D^2 - k^2)\right)\hat{\phi}^n \\ &+ a_n\hat{h}_v^n + b_n\hat{h}_v^{n-1}, \\ (D^2 - k^2)\hat{v}^{n+1} &= \hat{\phi}^{n+1}, \end{aligned} \quad (30)$$

where D signifies a derivative in the normal direction. Note that the above equations are two linear constant coefficient second order ordinary differential equations in y. A similar equation can also be derived from equation (28). These three equations can be written as follows,

$$(D^2 - \lambda^2)\hat{\phi}^{n+1} = \hat{f}_v^n, \quad (31)$$

$$(D^2 - k^2)\hat{v}^{n+1} = \hat{\phi}^{n+1}, \quad (32)$$

$$(D^2 - \lambda^2)\hat{\omega}^{n+1} = \hat{f}_\omega^n, \quad (33)$$

where

$$\lambda^2 = k^2 + 2R/(a_n + b_n), \quad (34)$$

$$\hat{f}_v^n = \hat{p}_v^n - \frac{2Ra_n}{a_n + b_n}\hat{h}_v^n, \quad (35)$$

$$\hat{f}_\omega^n = \hat{p}_\omega^n - \frac{2Ra_n}{a_n + b_n}\hat{h}_\omega^n, \quad (36)$$

and

$$\begin{aligned}\hat{p}_v^n &= -\left[D^2-\lambda^2+\frac{4R}{a_n+b_n}\right]\hat{\phi}^n-\frac{2Rb_n}{a_n+b_n}\hat{h}_v^{n-1}\\ &= -\hat{f}_v^{n-1}-\left[\frac{2R}{a_{n-1}+b_{n-1}}+\frac{2R}{a_n+b_n}\right]\hat{\phi}^n-\frac{2Rb_n}{a_n+b_n}\hat{h}_v^{n-1}, \qquad (37)\\ \hat{p}_\omega^n &= -\left[D^2-\lambda^2+\frac{4R}{a_n+b_n}\right]\hat{\omega}^n-\frac{2Rb_n}{a_n+b_n}\hat{h}_\omega^{n-1}\\ &= -\hat{f}_\omega^{n-1}-\left[\frac{2R}{a_{n-1}+b_{n-1}}+\frac{2R}{a_n+b_n}\right]\hat{\omega}^n-\frac{2Rb_n}{a_n+b_n}\hat{h}_\omega^{n-1}. \qquad (38)\end{aligned}$$

We will denote the quantities $\hat{p}_\omega^n$ and $\hat{p}_v^n$ the partial right hand sides of the equations.

2.6.2. *Horizontal velocities and wavenumber zero*

Having obtained $\hat{v}$ and $\hat{\omega}$ we can find $\hat{u}$ and $\hat{w}$ using equation (2) and the definition of the normal vorticity component, both transformed to Fourier space. We find

$$\hat{u} = \frac{i}{k^2}(\alpha D\hat{v}-\beta\hat{\omega}), \qquad (39)$$

$$\hat{w} = \frac{i}{k^2}(\alpha\hat{\omega}+\beta D\hat{v}). \qquad (40)$$

Similarly, we can find the streamwise and spanwise component of vorticity in terms of $\hat{\omega}$ and $\hat{\phi}$,

$$\hat{\chi} = \frac{i}{k^2}(\alpha D\hat{\omega}+\beta\hat{\phi}), \qquad (41)$$

$$\hat{\vartheta} = \frac{-i}{k^2}(\alpha\hat{\phi}+\beta D\hat{\omega}). \qquad (42)$$

These relations give the streamwise and spanwise components of velocity and vorticity for all wavenumber combinations, except when both α and β are equal to zero. In that case we have to use some other method to find $\hat{u}_0$, $\hat{w}_0$, $\hat{\chi}_0$ and $\hat{\vartheta}_0$ (the zero subscript indicates that $k=0$). The appropriate equations are found by taking the horizontal average of the first and the third component of equation (1). Due to the periodic BC all horizontal space derivatives cancel out, i.e.,

$$\frac{\partial u_0}{\partial t} = H_1+\frac{1}{R}\frac{\partial^2 u_0}{\partial y^2}, \qquad (43)$$

$$\frac{\partial w_0}{\partial t} = H_3+\frac{1}{R}\frac{\partial^2 w_0}{\partial y^2}. \qquad (44)$$

After a time discretization we find,

$$(D^2 - \lambda^2)\hat{u}_0^{n+1} = \hat{f}_{01}^n, \tag{45}$$

$$(D^2 - \lambda^2)\hat{w}_0^{n+1} = \hat{f}_{03}^n, \tag{46}$$

where

$$\hat{f}_{0i}^n = \hat{p}_{0i}^n - \frac{2Ra_n}{a_n + b_n}\hat{H}_{0i}^n, \tag{47}$$

and

$$\begin{aligned}\hat{p}_{0i}^n &= -\left(D^2 - \lambda^2 + \frac{4R}{a_n + b_n}\right)\hat{u}_{0i}^n - \frac{2Rb_n}{a_n + b_n}\hat{H}_{0i}^{n-1} \\ &= -\hat{f}_{0i}^{n-1}(\psi_0) - \left(\frac{2R}{a_{n-1} + b_{n-1}} + \frac{2R}{a_n + b_n}\right)\hat{u}_{0i}^n - \frac{2Rb_n}{a_n + b_n}\hat{H}_{0i}^{n-1}.\end{aligned} \tag{48}$$

Here the 0 index in $\hat{H}_{0i}$ refers to the zero wavenumber in both horizontal directions. Note that the above system contains the same type of equations as the system (32), and can thus be solved using the same numerical algorithm. Once $\hat{u}_0$ and $\hat{w}_0$ are calculated, the streamwise and spanwise components of vorticity for $k = 0$ can be found as follows,

$$\hat{\chi}_0 = D\hat{w}_0, \qquad \hat{\vartheta}_0 = -D\hat{u}_0. \tag{49}$$

2.6.3. *Solution procedure with boundary conditions*

A problem with the above equations is that the boundary conditions do not apply to the quantities for which we have differential equations. To remedy this, each of the equations can be solved for a particular solution with homogeneous boundary conditions. Then a number of homogeneous solutions with non-homogeneous boundary conditions are found for the same equations. Finally the boundary conditions are fulfilled by a suitable linear combination of particular and homogeneous solutions. Explicitly we proceed as follows:

For all $k = \sqrt{\alpha^2 + \beta^2} \neq 0$ and each of the two symmetries (symmetric and antisymmetric with respect to reflections around $y = y_L/2$) we solve :

$$(D^2 - \lambda^2)\hat{\phi}_p^{n+1} = \hat{f}_v^{n+1} \qquad \hat{\phi}_p^{n+1}(y_L) = 0 \tag{50}$$

$$(D^2 - k^2)\hat{v}_p^{n+1} = \hat{\phi}_p^{n+1} \qquad \hat{v}_p^{n+1}(y_L) = \begin{cases} \frac{\hat{v}_{BS}}{2} & \text{symmetric} \\ -\frac{\hat{v}_{BS}}{2} & \text{antisymmetric} \end{cases} \tag{51}$$

$$(D^2 - \lambda^2)\hat{\phi}_h^{n+1} = 0 \qquad \hat{\phi}_h^{n+1}(y_L) = 1 \tag{52}$$

$$(D^2 - k^2)\hat{v}_{ha}^{n+1} = \hat{\phi}_h^{n+1} \qquad \hat{v}_{ha}^{n+1}(y_L) = 0 \tag{53}$$

$$(D^2 - k^2)\hat{v}_{hb}^{n+1} = 0 \qquad \hat{v}_{hb}^{n+1}(y_L) = 1 \tag{54}$$

$$(D^2 - \lambda^2)\hat{\omega}_p^{n+1} = \hat{f}_\omega^{n+1} \qquad \hat{\omega}_p^{n+1}(y_L) = 0 \tag{55}$$

$$(D^2 - \lambda^2)\hat{\omega}_h^{n+1} = 0 \qquad \hat{\omega}_h^{n+1}(y_L) = 1, \tag{56}$$

where the subscripts p, h, ha and hb indicate the particular and the homogeneous parts. $\hat{v}_{BS}$ is only non-zero for cases with blowing and suction through the plate. Note that only one boundary condition is needed for each second order equation since the assumption of symmetry (or antisymmetry) takes care of the other. $\hat{v}_p^{n+1}(y_L) = 0$ when the symmetric and antisymmetric solutions are added and all the other solutions are zero at $y = 0$. Equations (52) and (56) have zero right hand sides and the same boundary conditions. The solution coefficients are therefore identical and so are also their symmetric and antisymmetric coefficients. Thus, four calls of the the equation solver can be reduced to one.

To fulfill the the remaining boundary conditions we first construct $\hat{v}_{p1}$, $\hat{v}_{h1}$ and $\hat{v}_{h2}$,

$$\hat{v}_{p1}^{n+1} = \hat{v}_p^{n+1} + C_{p1}\hat{v}_{ha}^{n+1} \quad \hat{v}_{p1}^{n+1}(y_L) = 0 \quad \hat{v}_{p1}^{n+1}(0) = v_{BS}/2 \tag{57}$$

$$\hat{v}_{h1}^{n+1} = \hat{v}_{ha}^{n+1} / \frac{\partial \hat{v}_{ha}}{\partial y}(y = y_L) \quad \hat{v}_{h1}^{n+1}(y_L) = 0 \quad \hat{v}_{h1}^{n+1}(0) = 0 \tag{58}$$

$$\hat{v}_{h2}^{n+1} = \hat{v}_{hb}^{n+1} + C_{h2}\hat{v}_{ha}^{n+1} \quad \hat{v}_{h2}^{n+1}(y_L) = 1 \quad \hat{v}_{h2}^{n+1}(0) = 0, \tag{59}$$

where C_{p1} and C_{h2} are chosen to fulfills the boundary condition $\partial v/\partial y = 0$ at the lower wall for each of the two symmetries of $\hat{v}_{p1}$ and $\hat{v}_{h2}$. As the symmetric and antisymmetric parts of $\partial \hat{v}_{h1}/\partial y$ cancel at the lower wall their sum v_{h1} fulfills $\partial v_{h1}/\partial y = 0$.

Now the solutions (v_{p1}, ω_p), $(v_{h1}, \omega = 0)$, $(v_{h2}, \omega = 0)$ and $(v = 0, \omega_h)$ fulfill all the physical boundary conditions at the lower wall. The total normal velocity and vorticity is then given by

$$\hat{v}^{n+1} = \hat{v}_{p1}^{n+1} + C_{v1}\hat{v}_{h1}^{n+1} + C_{v2}\hat{v}_{h2}^{n+1}, \tag{60}$$

$$\hat{\omega}^{n+1} = \hat{\omega}_p^{n+1} + C_\omega \hat{\omega}_h^{n+1}, \tag{61}$$

where C_{v1},C_{v2} and C_ω are chosen such that the boundary conditions at the upper boundary are fulfilled. The u and w velocities are found from the definition of the normal vorticity and the incompressibility constraint.

In general we have to find u and w first to evaluate the boundary conditions. Thus with the C's unknown we find :

$$\hat{u}^{n+1} = \hat{u}_{p1}^{n+1} + C_{v1}\hat{u}_{h1}^{n+1} + C_{v2}\hat{u}_{h2}^{n+1} + C_\omega \hat{u}_h^{n+1}, \tag{62}$$

$$\hat{w}^{n+1} = \hat{w}_{p1}^{n+1} + C_{v1}\hat{w}_{h1}^{n+1} + C_{v2}\hat{w}_{h2}^{n+1} + C_\omega \hat{w}_h^{n+1}, \tag{63}$$

where (u_{p1}, w_{p1}), (u_{h1}, w_{h1}), (u_{h2}, w_{h2}) and (u_h, w_h) are found from (v_{p1}, ω_p), $(v_{h1}, \omega = 0)$, $(v_{h2}, \omega = 0)$ and $(v = 0, \omega_h)$ using equation (39) and (40).

Assuming the boundary conditions are linear we can write them as :

$$L_i(\hat{u}, \hat{v}, \hat{w}) = \hat{D}_i; \qquad i = 1, 2, 3. \tag{64}$$

Here L_i is the linear operator for the ith boundary condition. This can include derivatives in the wall normal direction. The operator may also depend on the wave number (for example when the boundary condition contains horizontal derivatives). Note that the expression for evaluation L_i may include $\hat{\omega}$ as this is equivalent to horizontal derivatives. $\hat{D}_i$ is the data for the boundary condition, the most common form of which is is either zero (homogeneous boundary conditions) or the operator L_i applied to a base flow.

Finally inserting the expressions (60), (62) and (63) into equation (64) and moving all terms containing the particular solution to the right hand side, we get a three by three linear system of equations which is easily solved to find the C's.

For $k = 0$ we solve

$$(D^2 - \lambda^2)\hat{u}_{p0}^{n+1} = \hat{f}_{01}^n \qquad \hat{u}_{p0}^{n+1}(0) = u_{low};\ \hat{u}_{p0}^{n+1}(y_L) = u_{upp} \tag{65}$$

$$(D^2 - \lambda^2)\hat{w}_{p0}^{n+1} = \hat{f}_{03}^n \qquad \hat{w}_{p0}^{n+1}(0) = w_{low};\ \hat{w}_{p0}^{n+1}(y_L) = w_{upp} \tag{66}$$

$$(D^2 - \lambda^2)\hat{u}_{h0}^{n+1} = 0 \qquad \hat{u}_{h0}^{n+1}(0) = 0;\ \hat{u}_{h0}^{n+1}(y_L) = 2 \tag{67}$$

$$(D^2 - \lambda^2)\hat{w}_{h0}^{n+1} = 0 \qquad \hat{w}_{h0}^{n+1}(0) = 0;\ \hat{w}_{h0}^{n+1}(y_L) = 2, \tag{68}$$

where u_{low}, u_{upp}, w_{low} and w_{upp} denote the lower and upper wall velocities. Computations in a moving reference frame can increase the time step. If the boundary condition at the upper wall is in the form of Dirichlet type (specified velocity) then

$$\hat{u}_0 = \hat{u}_{p0}, \tag{69}$$

$$\hat{w}_0 = \hat{w}_{p0}. \tag{70}$$

For other types of upper wall boundary conditions we find the complete solution from :

$$\hat{u}_0 = \hat{u}_{p0} + C_u \hat{u}_{h0}, \tag{71}$$

$$\hat{w}_0 = \hat{w}_{p0} + C_w \hat{w}_{h0}, \tag{72}$$

where C_u and C_w are chosen so that $\hat{u}_0$ and $\hat{w}_0$ fulfill the boundary conditions.

The above equations are all in Fourier space, where the non-linear terms h_v, h_ω, H_1 and H_3 become convolution sums. These sums can be efficiently calculated by transforming the velocities and vorticities using FFTs to physical space, where they are evaluated using pointwise products.

2.7. *Normal discretization – Chebyshev expansion*

The typical equation derived above is a second order constant coefficient ODE of the form

$$(D^2 - \kappa)\hat{\psi} = \hat{f} \qquad \hat{\psi}(0) = \gamma_{-1}, \quad \hat{\psi}(y_L) = \gamma_1. \tag{73}$$

First map the interval $[0, y_l]$ to $[-1, 1]$ by setting $y' = 2y/y_L - 1$. Then

$$(D'^2 - \nu)\hat{\psi} = \hat{f} \qquad \hat{\psi}(-1) = \gamma_{-1}, \quad \hat{\psi}(1) = \gamma_1, \tag{74}$$

where $\nu = \kappa y_L^2/4$. In the following we have for simplicity dropped the prime.

This equation can be solved accurately if the dependent variable $\hat{\psi}$, its second derivatives, the right hand side $\hat{f}$ and the boundary conditions are expanded in Chebyshev series, i.e.,

$$\hat{\psi}(y) = \sum_{j=0}^{N_y} \tilde{\psi}_j T_j(y), \tag{75}$$

$$D^2\hat{\psi}(y) = \sum_{j=0}^{N_y} \tilde{\psi}_j^{(2)} T_j(y), \tag{76}$$

$$\hat{f}(y) = \sum_{j=0}^{N_y} \tilde{f}_j T_j(y), \tag{77}$$

$$\hat{\psi}(1) = \sum_{j=0}^{N_y} \tilde{\psi}_j = \gamma_1, \tag{78}$$

$$\hat{\psi}(-1) = \sum_{j=0}^{N_y} \tilde{\psi}_j (-1)^j = \gamma_{-1}, \tag{79}$$

where T_j are the Chebyshev polynomial of order j and N_y the highest order of polynomial included in the expansion. If the Chebyshev expansions are used

in equation (74), together with the orthogonality properties of the Chebyshev polynomials, we find the following relation between the coefficients

$$\tilde{\psi}_j^{(2)} - \nu\tilde{\psi}_j = \tilde{f}_j \qquad\qquad j = 0, ...N_y. \tag{80}$$

By writing the Chebyshev functions as cosines and using well known trigonometric identities, one finds relations between the Chebyshev coefficients of $\hat{\psi}$ and those of its derivative that can be used for differentiation and integration (see Canuto *et al.* (1988))

$$\tilde{\psi}_j^{(p)} = \sum_{\substack{m=j+1 \\ m+j\ odd}}^{N_y} m\tilde{\psi}_m^{(p-1)} \qquad\qquad j = 1, ...N_y, \tag{81}$$

$$\tilde{\psi}_j^{(p-1)} = \frac{1}{2j}(c_{j-1}\tilde{\psi}_{j-1}^{(p)} - \tilde{\psi}_{j+1}^{(p)}) \qquad\qquad j = 1, ...N_y, \tag{82}$$

where the superscript p indicates the order of the derivative and $c_j = 2$ for $j = 0$ and $c_j = 1$ for $j > 0$. In the first differentiation relation one observes that an error in the highest order coefficients of $\tilde{\psi}^{(p-1)}$ influences all coefficients of its derivative $\tilde{\psi}^{(p)}$. This problem is what is supposed to be avoided by the Chebyshev integration method discussed below. In the second relation we assume that $\tilde{\psi}_j^{(p)} = 0$ for $j > N_y$ and note that $\tilde{\psi}_0^{(p-1)}$ is an integration constant needed when the function $\hat{\psi}^{(p-1)}$ is found by integrating $\hat{\psi}^{(p)}$. Note also that the integration procedure introduces a truncation error, since an integration of a Chebyshev polynomial would result in a polynomial of one degree higher. The coefficient $\tilde{\psi}_{N_y+1}^{(p-1)}$ which would have multiplied T_{N_y+1} is in the present truncation set to zero.

If the relations (82) are used together with relation (80) a system of equations can be derived for either coefficients $\tilde{\psi}_j$ or $\tilde{\psi}_j^{(2)}$. The second approach, called the Chebyshev integration method (CIM), was proposed by Greengard (1991) to avoid the ill conditioned process of numerical differentiation in Chebyshev space. It was implemented in the original channel code by Lundbladh *et al.* (1992*a*) and is also included in the present implementation. However, we have found that using this method, subtle numerical instabilities occur in some cases and we therefore recommend to solve for the coefficients of the function itself, $\tilde{\psi}_j$. Such a Chebyshev tau method (CTM), almost identical to that used by Kim, Moin & Moser, is also implemented and is so far found to be stable. We first present the CTM, then the CIM and finally we discuss the instabilities observed in computations with the CIM. Note that the instabilities have occurred only a few times and that the results otherwise are the same for the two methods.

2.7.1. *Chebyshev tau method-CTM*

If the recursion relation (82) is used to express equations (80) in the coefficients $\tilde{\psi}_j$, one arrives at the system of equations (83 below). A more detailed derivation can be found in Canuto *et al.* (1988), but observe the sign errors therein. We have

$$-\frac{c_{j-2}\nu}{4j(j-1)}\tilde{\psi}_{j-2}+\left(1+\frac{\nu\beta_j}{2(j^2-1)}\right)\tilde{\psi}_j-\frac{\nu}{4j(j+1)}\tilde{\psi}_{j+2}$$
$$=\frac{c_{j-2}}{4j(j-1)}\tilde{f}_{j-2}-\frac{\beta_j}{2(j^2-1)}\tilde{f}_j+\frac{\beta_{j+2}}{4j(j+1)}\tilde{f}_{j+2},\quad j=2,\ldots,N_y \tag{83}$$

where

$$\beta_j=\begin{cases}1 & 0\le j\le N_y-2\\ 0 & j>N_y-2\end{cases}. \tag{84}$$

Note that the even and odd coefficients are uncoupled. Since a Chebyshev polynomial with an odd index is an odd function, and vice versa, the decoupling of the systems of equations is just a result of the odd and even decoupling of equation (74) itself. The same can be achieved for the boundary conditions (78) and (79) if they are added and subtracted,

$$\sum_{\substack{j=0\\ j\ even}}^{N_y}\tilde{\psi}_j=\frac{\gamma+\gamma_-}{2},\qquad \sum_{\substack{j=1\\ j\ odd}}^{N_y}\tilde{\psi}_j=\frac{\gamma-\gamma_-}{2}. \tag{85}$$

These boundary conditions together with the equations (83) constitute a linear system of N_y+1 equations that can be solved for the coefficients $\tilde{\psi}_j$ ($j=0,\ldots,N_y$). The structure of the equations involving the even coefficients forms a tridiagonal system and so does the equation for the odd coefficients. The boundary conditions fill the top row of both systems and make the systems only quasi-tridiagonal, but it only takes $16N_y$ operations to solve both systems.

The system (83) has in fact been truncated to only contains N_y-1 equations and two equations have been replaced by boundary conditions. That truncation introduces what is usually called the tau error. In solution algorithms that solve for the three velocity components of the Navier-Stokes equations and the pressure, the coupling between the equations for the velocities and that for the pressure requires corrections of the tau error (Kleiser & Schumann 1980). We have chosen to eliminate the pressure in the Navier-Stokes equations and solve for the normal velocity and the normal vorticity. As those equations do not couple in the same way, we do not have to correct the tau error.

2.7.2. *Chebyshev integration method-CIM*

Instead of solving for the coefficients $\tilde{\psi}_j$, the CIM solves for the coefficients of the Chebyshev series for the second derivative, $\tilde{\psi}_j^{(2)}$. The major advantage is

supposed to come in the calculation of derivatives of the solution $\hat{\psi}$. Derivatives are needed in the calculation of the remaining velocities and vorticities using equations (39)-(42). In the CIM the second derivative is already calculated and the first derivative and the function itself can be found by the numerically well conditioned process of integration.

If the relations (82) are used to write (80) in terms of $\tilde{\psi}_j^{(2)}$ the result is the following system of equations,

$$\begin{aligned}
j=0: &\quad \tilde{\psi}_0^{(2)} - \nu\tilde{\psi}_0 &&= \tilde{f}_0 \\
j=1: &\quad \tilde{\psi}_1^{(2)} - \nu(\tilde{\psi}_0^{(1)} - \tfrac{1}{8}\tilde{\psi}_1^{(2)} + \tfrac{1}{8}\tilde{\psi}_3^{(2)} &&= \tilde{f}_1 \\
2\le j\le N_y-2: &\quad \tilde{\psi}_j^{(2)} - \nu\tfrac{1}{4j}\left[\tfrac{c_{j-2}\tilde{\psi}_{j-2}^{(2)}}{j-1} - \tilde{\psi}_j^{(2)}\left(\tfrac{1}{j-1}+\tfrac{1}{j+1}\right) + \tfrac{\tilde{\psi}_{j+2}^{(2)}}{j+1}\right] &&= \tilde{f}_j \quad (86) \\
j=N_y-1: &\quad \tilde{\psi}_{N_y-1}^{(2)} - \nu\tfrac{1}{4(N_y-1)}\left[\tfrac{\tilde{\psi}_{N_y-3}^{(2)}}{N_y-2} - \tilde{\psi}_{N_y-1}^{(2)}\left(\tfrac{1}{N_y-2}+\tfrac{1}{N_y}\right)\right] &&= \tilde{f}_{N_y-1} \\
j=N_y: &\quad \tilde{\psi}_{N_y}^{(2)} - \nu\tfrac{1}{4N_y(N_y-1)}(\tilde{\psi}_{N_y-2}^{(2)} - \tilde{\psi}_{N_y}^{(2)}) &&= \tilde{f}_{N_y}.
\end{aligned}$$

The equations for odd and even coefficients decouple and so do the boundary conditions on the form (85). However, we now need to rewrite them with the aid of (80) to contain the coefficients of $\tilde{\psi}^{(2)}$ that we are now solving for. We find that the first sum in (85) takes the form,

$$\begin{aligned}
\tilde{\psi}_0 + \tilde{\psi}_0^{(1)} \quad &+\tfrac{1}{4}\tilde{\psi}_0^{(2)} - \tfrac{1}{12}\tilde{\psi}_1^{(2)} - \tfrac{7}{48}\tilde{\psi}_2^{(2)} + \textstyle\sum_{j=3}^{N_y-2}\frac{3\tilde{\psi}_j^{(2)}}{(j-2)(j-1)(j+1)(j+2)} \\
&-\tfrac{(N_y-6)\tilde{\psi}_{N_y-1}^{(2)}}{4(N_y-3)(N_y-2)N_y} - \tfrac{\tilde{\psi}_{N_y}^{(2)}}{2(N_y-2)(N_y-1)N_y} = \gamma_1. \qquad (87)
\end{aligned}$$

Thus, the solution of equation (74) is found by solving the system of equations for the second derivative of $\tilde{\psi}$ (87) together with the boundary conditions (87) and the corresponding one at $y=-1$. We now have two more equations than for the tau method and the solution to the full system is a set of N_y+1 coefficients of the second derivative and the two integration constants $\tilde{\psi}_0^{(1)}$ and $\tilde{\psi}_0^{(2)}$ representing the zeroth order Chebyshev coefficient of $D\hat{\psi}$ and $\hat{\psi}$ itself, respectively. The function $\hat{\psi}$ is then found by two integrations, which in Chebyshev space can easily be constructed using the relations (82). The same quasi-tridiagonal form of the equation systems for the odd and even coefficients appears as for the CTM and the same solution routine can be used.

2.7.3. *Integration correction*

When the solution for $\hat{\psi}^{(2)}$ is found by the CIM and integrated to obtain $\hat{\psi}^{(1)}$ and $\hat{\psi}$ the same truncation is used for both the derivatives and $\hat{\psi}$ itself. They are all represented with N_y+1 non-zero Chebyshev coefficients. This means

that the truncations are not compatible, since the derivative of a function represented as a finite Chebyshev series should have one coefficient less than the function itself. For example, if the coefficients $\tilde{\psi}_j$ are used to construct those for the derivative, using the recurrence relation (81), the result will not be the same as the coefficients $\tilde{\psi}_j^{(1)}$. There will be a slight difference in half of the coefficients for the derivative, the size depending on the magnitude of the coefficient $\tilde{\psi}_{N_y}$. The expression for the difference can be derived as follows. We write $\hat{\psi}$ explicitly using the coefficients $\tilde{\psi}_j^{(1)}$ and the relation (82),

$$\hat{\psi} = \tilde{\psi}_0 T_0 + \sum_{j=1}^{N_y-1} \frac{1}{2j}(c_{j-1}\tilde{\psi}_{j-1}^{(1)} - \tilde{\psi}_{j+1}^{(1)})T_j + \frac{1}{2N_y}\tilde{\psi}_{N_y-1}^{(1)} T_{N_y}. \tag{88}$$

Now (81) is applied to the Chebyshev coefficients in (88) to calculate the derivative $D\hat{\psi}$. Let $\tilde{\psi}_j^D$ be its new coefficients. We find that these new coefficients will not equal $\tilde{\psi}_j^{(1)}$ and the following relation is found between them,

$$\begin{aligned} \tilde{\psi}_j^D &= \tfrac{2}{c_j} \textstyle\sum_{\substack{q=j+1 \\ q+j\ odd}}^{N_y} (c_{q-1}\tilde{\psi}_{q-1}^{(1)} - \tilde{\psi}_{q+1}^{(1)}) + \quad \frac{1}{c_j}\tilde{\psi}_{N_y-1}^{(1)} \\ &= \tilde{\psi}_j^{(1)} \qquad\qquad q + N_y\ odd, \end{aligned} \tag{89}$$

$$\begin{aligned} \tilde{\psi}_j^D &= \tfrac{2}{c_j} \textstyle\sum_{\substack{q=j+1 \\ q+j\ odd}}^{N_y} (c_{q-1}\tilde{\psi}_{q-1}^{(1)} - \tilde{\psi}_{q+1}^{(1)}) \\ &= \tilde{\psi}_j^{(1)} - \tfrac{1}{c_j}\tilde{\psi}_{N_y}^{(1)} \qquad\qquad q + N_y\ even. \end{aligned} \tag{90}$$

Thus, we have a method of correcting the coefficients $\tilde{\psi}_j^{(1)}$ so that they represent $D\hat{\psi}$ with the same truncation as $\tilde{\psi}_j$ represent $\hat{\psi}$. A similar correction can be derived for the coefficients $\tilde{\psi}_j^{(2)}$ of the second derivative. After some algebra we find,

$$\tilde{\psi}_j^{D^2} = \tilde{\psi}_j^{(2)} - \frac{1}{c_j}\left(1 + \frac{(N_y-1)^2 - j^2}{4N_y}\right)\tilde{\psi}_{N_y-1}^{(2)} \qquad j + N_y\ odd, \tag{91}$$

$$\tilde{\psi}_j^{D^2} = \tilde{\psi}_j^{(2)} - \frac{1}{c_j}\tilde{\psi}_{N_y}^{(2)} \qquad j + N_y\ even, \tag{92}$$

where $\tilde{\psi}_j^{D^2}$ are the corrected Chebyshev coefficients for $D^2\hat{\psi}$.

When the horizontal components of velocity and vorticity are found using the relations (39) to (42), we need $\hat{\phi}$, $D\hat{v}$ and $D\hat{\omega}$. The above corrections are therefore needed in order for the velocity and vorticity fields to exactly satisfy the incompressibility constraint (2). Note that an error in the highest Chebyshev coefficients will by the above correction scheme affect all other coefficients of the first and second derivative. Exactly what was supposed to be avoided by the integration method.

The CTM and CIM methods are equally efficient and give the same results with the exception of a few very rare cases. We have found that numerical instabilities may occur when the wall normal resolution is very low and the velocity and vorticity fields are not divergence free. We have also found that it in those cases is enough to make the vorticity divergence free to stabilize the calculations. With integration correction or the CTM method, both velocity and vorticity are completely divergence free. However, for one channel flow case so far, and more frequently in the boundary layer, a numerical instability occurs with the integration correction but not without.

Fortunately the instability causes the calculation to blow up in a few time steps and before that the results are the same as for a stable version of the code. With sufficient wall normal resolution (which is required anyhow) and without the integration correction the boundary layer code has been found completely reliable. The CTM method is, however, to prefer.

2.8. *Pressure*

By expressing the Navier-Stokes equations in the form of equations (4) and (7), the pressure need not to be taken into account. However, it might be of interest to solve for this quantity as well as the velocity components. The pressure can, for example, be used for detecting regions of rapid motion in a turbulent boundary layer.

The Poisson equation for the pressure derived above, equation (3), is written as

$$\nabla^2(p+E) = \frac{\partial H_i}{\partial x_i}, \tag{93}$$

where $E = \frac{1}{2}u_i u_i$ and $H_i = h_i + F_i = \epsilon_{ijk}u_j(\omega_k + 2\Omega_k) + F_i$. Note that the term F_i does not contain the disturbances in the fringe region for the spatial simulations and is zero for the temporal boundary layer. This equation has a similar form as the equations for ϕ, v and ω and can thus be solved using the same numerical routine.

The boundary conditions at the wall ($y = 0$) and at the upper boundary ($y = y_L$) are derived from the normal component of the Navier-Stokes equations. The boundary condition with non-zero wall velocities becomes

$$\left.\frac{\partial}{\partial y}(p+E)\right|_{y=0} = \left.\left[\frac{1}{R}\nabla^2 v + h_2 - \frac{\partial v}{\partial t}\right]\right|_{y=0}. \tag{94}$$

The term $\frac{\partial v}{\partial t}$ is included for the case of flow control like blowing/suction from the wall. For a wall with zero velocities the boundary condition becomes

$$\left.\frac{\partial}{\partial y}(p+E)\right|_{y=0} = \left.\frac{1}{R}\frac{\partial^2 v}{\partial y^2}\right|_{y=0}. \tag{95}$$

At $y = y_L$ the boundary condition becomes

$$\frac{\partial}{\partial y}(p+E)\bigg|_{y=y_L} = \left[\frac{1}{R}\nabla^2 v + h_2 + \lambda(x)(\mathcal{U}_y - v) - \frac{\partial v}{\partial t}\right]\bigg|_{y=y_L}, \qquad (96)$$

where $\lambda(x)$ is the fringe function described in section 2.4.

For wavenumber zero the boundary condition (96) is automatically fulfilled if boundary condition (94) is fulfilled. It is required by the compatibility condition

$$\int_0^{y_L} \frac{dH_2}{dy} dy = \frac{\partial}{\partial y}(p+E)|_{y=y_L} - \frac{\partial}{\partial y}(p+E)|_{y=0}, \qquad (97)$$

which comes from the integration of equation (93). A second boundary condition for p itself is needed at $y = 0$ and this is chosen to be $p = 0$. The mean pressure at the wall cannot be determined and $p = 0$ at the wall is a reference pressure. It is not possible to choose $p = 0$ at $y = y_L$ because the location of the free-stream is arbitrary chosen for numerical purposes.

It might seem to be a better approach to rewrite equation (3) as

$$\nabla^2 p = -\frac{\partial u_i}{\partial x_j}\frac{\partial u_j}{\partial x_i} + \frac{\partial}{\partial x_i}(2\epsilon_{ijk}u_j\Omega_k) + \frac{\partial F_i}{\partial x_i}, \qquad (98)$$

and solve for the pressure directly. The solution to equation (98) turns out to be sensitive to the values of the velocities at the upper boundary. When using different boundary conditions for the velocities, the solutions are slightly different, hence the pressure will be different. The sensitivity comes from the fact that derivation in the normal direction in Chebyshev space is dependent on the coefficients in all the collocation points. These coefficients change when transforming back and forth to physical space. Thus the derivations must be, for consistency, performed at the same time, with no transformations between them. These problems are avoided by solving for the pressure plus energy as in equation (93).

The pressure can be calculated from a specific velocity field with the post processing program **pre**. The pressure needs thus not be calculated in the simulation itself. If turbulent statistics involving pressure are being calculated during a simulation, the pressure is calculated in those time steps where the sampling occurs.

3. Implementation

In implementing the algorithm presented above a significant effort has been put into portability, flexibility and computational efficiency. The language is standard FORTRAN 77 with the extension of the INCLUDE statements, eight character names and lower case characters. Especially the demands on the data structure have forced an encapsulation of the access to the main storage which requires some attention. Also the vectorization and the need to process suitably large chunks of data at a time adds complexity in exchange for execution speed.

3.1. *Program structure of* **bla**

The program **bla** has been divided into subroutines each with one specific task. The main program steps the time and calculates the adaptive time step. The subroutines **nonlinbl** and **linearbl** carry out the main part of the algorithm aided by smaller subroutines for integration, equation solving etc. The FFTs are taken from **VECFFT** which was developed specifically for the simulation codes but is an independent package of vectorizable Fourier and Chebyshev transforms.

3.1.1. *Coarse program structure, step 1 - 4*

Since some computers cannot hold all of the three dimensional data in the main memory simultaneously, and in any case the number of three dimensional arrays should be minimized to save space, the three dimensional computation is carried out by slicing the data into two dimensional planes.

In the main time stepping loop the data needs to be stepped through twice. First slicing in x-z-planes to calculate the FFTs and the pointwise product for non-linear terms, step 2, and second in x-y-planes to calculate the normal Chebyshev transforms and solve the equation systems for the new velocities and vorticities, step 3. Step 1 reads input files, initializes the FFTs and calculates the partial right hand sides needed to start the time stepping loop and computes the base flow. Step 4 stores the final velocity field.

3.1.2. *Step 1, initialization*

Subroutine **ppar** prints the contents of the parameter file to standard output as a check of which size of problem the image is compiled for.

Subroutine **rparambl** reads the file **bla.i** which contains control information for the program, especially the input and output filenames and the final time to which the simulation is to be done, cf. section 5.2.

Subroutine **rdiscbl** reads the resolution, the computational box size and a few parameters defining the flow from the file **namnin**. The velocities are then read from the file and put into the main storage positions 1-3. If the resolution of the image and the file do not correspond, this is printed on standard output and the program stops execution. The check can be disabled by the `varsiz` flag in the **bla.i** file in which case the field is extended by zero-padding or truncated to fit the image resolution.

Subroutine **rescale** rescales all data read from **bla.i** from boundary layer scaling to the channel flow scaling used internally, see Appendix B.

Subroutine **fskch** computes the base flow boundary layer profile.

Subroutine **preprbl** calculates wavenumbers and collocation points, and initializes the FFTs.

Subroutine **fshift** computes a Galilei transformation which can be used to increase the maximum stable time step.

Subroutine **rwavebl** reads the profile of forcing waves to be introduced in the fringe region.

Subroutine **getdt** calculates the initial time step to set get a CFL number equal to the `cflmax` value. The subroutine is only used if the time stepping is adaptive.

Subroutine **prhs** calculates the initial partial right hand sides $\hat{p}_\phi$ $\hat{p}_\omega$, $\hat{p}_{01}$, $\hat{p}_{03}$ and places the first two in positions 6 and 7 of the main storage. The streamwise and spanwise vorticities are also calculated and put into positions 4 and 5 of the storage.

Subroutine **bflow** generates a base flow used for spatial simulations.

Subroutine **cbflow** reads or writes the base flow boundary layer profile in **basic.i** for spatial simulations.

Subroutine **blfou** computes the streamwise Fourier transform of the base flow.

Some initial parameter values for the time stepping mechanism are prepared in the main program and output files are opened.

3.1.3. *Step 2, computations in physical space*

The subroutine **wplbl** writes data to 2-d plane files.

The subroutine **blshift** shifts the base flow and boundary conditions to be aligned with the computational domain when a Galilean transform is used, i.e. if the lower wall is "moving".

The subroutine **gtrip** generates a random force flow trip.

The subroutine **boxxys** computes the spanwise and time averaged statistics for one xz-box.

The subroutine **nonlinbl** calculates H_i as pointwise products in physical space and stores them in position 1 to 3 of the main storage. It also computes the volume forcing and adds it to H_i. As the main storage is in Fourier-physical space, cf. section 3.2.2 below, the velocities and vorticities must be transformed back to physical space before the product can be formed. Likewise the products H_i must be transformed to Fourier space before storing them. The velocity rms amplitudes are computed in Fourier-physical space. The maximum CFL number and the extrema of the velocities are calculated from the velocities in physical space.

The xy-statistics, CFL number, and rms-amplitude and extremum statistics are written to the respective files.

3.1.4. *Step 3, computations in Fourier-Chebyshev space*

The time step is recalculated to regulate the CFL number close to `cflmax` if adaptive time stepping is enabled. The time stepping parameters are calculated for the next time step.

Subroutine **linearbl** transforms the non-linear products into Chebyshev space and constructs the complete right hand sides for the evolution equations. The Chebyshev-tau or Chebyshev-integration method is used to solve for the evolution variables from a set of tridiagonal equations. The chosen boundary conditions are applied. All velocities and vorticities are constructed and partial right hand sides are computed for the next time step. Finally the velocities and vorticities are transformed back to physical space in the y-direction. The velocities are stored into positions 1 to 3, the streamwise and spanwise vorticity into 4 and 5 and the partial right hand sides into 6 and 7 of the main storage.

For selected times the 3-dimensional velocity data is written to file.

Time is incremented and execution is continued with the next time step from step 2 if the the final time `tmax` is not reached.

If `pressure` is set to one in the file **par.f**, the following two subroutines are entered if statistics are sampled in this step. In **nonlinp** the terms $H_{1,1} + H_{3,3}$ and H_2 are calculated and stored in position 4 and 5. The energy E is calculated and stored in position 8. In **linearp** the linear and non-linear parts of the boundary conditions and the sum $H_{1,1} + H_{2,2} + H_{3,3}$ are calculated. The equation for the pressure is solved and the streamwise and spanwise vorticity need to be recalculated. Pressure is stored in position 8.

3.1.5. *Step 4, output*

The subroutine **wdiscbl** handles the output of a velocity field to an external file. The final values of xy-statistics are written to file by **wxys**. The pressure is written to an external file by **wdiscp** if `pressure` is set to one. The amplitude files are written by **wamp**, and planes are written by **wplbl**. All opened files are closed.

3.2. *Data structure*

As the size of a problem is explicitly compiled into the program, the memory allocation is for the most part static. Some effort was put into minimizing not only the three dimensional storage but also the two dimensional arrays

since this is the only part residing in main memory when the three dimensional storage is located on an external device.

3.2.1. *Complex numbers and FFTs*

Most of the algorithm above works with quantities in Fourier space. These are in general complex which requires storage of both real and imaginary parts. Though FORTRAN has the capability of automatically handling complex numbers most compilers produce inefficient code for this, especially for mixed real and complex expressions. Moreover FORTRAN stores complex numbers with alternating real and imaginary parts, which causes a severe performance loss for vector fetches on certain computers as the stride will be even. To circumvent this, it was decided to store all complex quantities in double arrays, one for real and one for imaginary parts. As the algorithm neither includes general complex-complex multiplications nor divisions this did not add very much code.

The FFTs in **VECFFT** are built for separate storage of the real and imaginary parts, but can optionally be used with standard FORTRAN storage.

3.2.2. *Main storage, boxes, drawers, and planes*

As mentioned above, to save on space the algorithm traverses the three dimensional volume twice to complete a time step. The three dimensional storage is in some cases too large to fit in the main memory in which case it may be put on an external device such as an SSD or a disk. In order to efficiently access this external device the records need to be long, preferably much longer than the typical vector length needed to get good CPU performance. If the three dimensional storage is divided into x-z- and x-y-planes the largest common element between these is a single vector in the x-direction, a *pencil* containing `nx` words. In order to increase this number, planes are combined into a *box* consisting of an integer number of adjacent planes e.g., an x-y-box holds `mbz` x-y-planes and an x-z-box holds `mby` x-z-planes. The intersection between an x-y- and an x-z-box then holds `mby*mbz` pencils, which is called a *drawer*. Most subroutines are made to handle a box rather than a plane at a time, with the additional advantage that the vector length increases by a factor of `mbz` or `mby`.

The variables in the main storage are in Fourier-physical format, i.e., the axes are α, physical y and β, except for the partial right hand sides $\hat{p}_v$ and $\hat{p}_\omega$, which are stored in Fourier-Chebyshev space.

The structure of the file used for the three dimensional storage is as follows: File format : unformatted, direct access, scratch, record length `nx*mby*mbz*npreal` bytes, name **ur**. `npreal` is the number of bytes used to store a real number (usually 4 or 8 bytes). Storage sequence: the drawers are stored in increasing `y, z` and `i` order, with `y` varying the fastest and `i` slowest. Within each drawer

the coefficients are stored in increasing `x, y, z` order with `x` varying the fastest and `z` slowest. All the real data is stored in the first half of the drawer and imaginary data in the second. The number of records is `nby*nbz*7`.

The main storage is accessed box-wise by the routines **getxy, putxy, getxz** and **putxz**. The routines select between core storage and file storage depending on the value of the integer `nfc` (1/0); for the latter case the routines **getdr, putdr** move one drawer from or to the file.

3.2.3. *Naming conventions*

The variable names in the algorithm description above have been followed as closely as possible. One important exception is that N_y in the algorithm corresponds to `ny-1`. Greek letters have been replaced by abbreviations. In the case a variable is complex it has been replaced by two with the last letters 'r' and 'i', for the real and imaginary parts. An example of this is `pomyr` which is the real part of the array $\hat{p}^n_\omega$. Note that the superscripts 'n' etc. and the hat symbol are generally left out, when needed for distinction they are replaced by suffices , e.g. a^{n+1} becomes `anp1`. The component indices '1,2,3' in, e.g., H_1 are usually found as the last index of the array. Instead numbers in the array names are used to distinguish between the same variable when represented by two different arrays in step 2 and step 3. Normal derivatives are denoted by prefixes `d` and `d2`. Sometimes a 'b' is used for 'box', cf. above, e.g., `bbeta` is the wavenumber beta vector expanded to correspond to other box sized arrays.

All variables are declared a specific type and the program has been compiled with an `implicit none` statement, which was changed to `implicit logical (a-z)` as the former is non-standard. Thus the type rules are not into effect and have not been adhered to; note especially that `x`, `y`, `z` are integer indices in do loops.

4. Operation

The program **bla** reads a velocity field from an external device, steps the field to a selected final time while producing some log information on the standard output device and writes the final velocities back to a file. During the simulation it may also output a file of the velocity and vorticity rms amplitudes, a file of the amplitude of specific wavenumbers, a file of extremum amplitudes, a file of statistics averaged over the spanwise direction, files with velocities in two dimensional planes at regular intervals in time and files containing complete 3-d velocity fields at selected times. The simulation can be run with the pressure solver to get the pressure at the same time steps as the velocities.

The program **bls** may be used to produce the initial velocity field.

The program **rit** performs post processing of 3-d velocity fields into Tektronix, Postscript or ppm (portable pixel map) compatible graphics. Linear combinations (for example difference) of one or more 3-d velocity fields can be computed with **cmp**, which also can calculate rms and maxnorm amplitudes of the result. This is useful for, for example, convergence checks.

The program **pre** calculates the pressure for a 3-d velocity field and produces a 3-d pressure field which is post processed with **ritpre**.

Postprocessing of two dimensional planes is done by the program **rps** in a way similar to **rit**. Plots of amplitude files are generated by the programs **pamp1** and **pamp2**, which handle one and multiple amplitude files respectively. Wave amplitude files are plotted by the program **pampw** and **pampw2** and extremum amplitude files by **pext1**.

To reduce the storage requirements of 3-d velocity files, they can be compressed by **dfc** and similarly for two dimensional plane files by **dpc**. Note that regular compression programs such as gzip or compress give a negligible reduction in size of these binary data files. An additional advantage with using the compression routines is that they produce a binary data format which is portable between machines with different file formats and floating point representations.

These programs along with the Fourier transform library **VECFFT**, the compression library **dclib** and the plot library **plot1** forms a completely self contained and portable system written in FORTRAN 77.

4.1. *Compiling*

Most of the programs need to be recompiled for each size of problem to be run. Under UNIX this is most easily handled with a makefile. As stated above the compiler must handle INCLUDE statements and lower case characters. For compilation most of the programs require Fourier transforms from the package **VECFFT**. These are also written in standard FORTRAN 77 and can be compiled along with the code. The number of grid points and some other parameters must be set prior to compilation in the file **par.f**. The same **par.f** file should be used for the compilation of all programs to work on a specific simulation. Which routines that need to be recompiled after changing the parameter file is determined by the makefiles.

The number of spectral modes in each direction is set by the parameters `nx`, `ny`, `nz`. The following restrictions apply : `nx` and `ny-1` must be even and factorable by 2, 3 and 5, `nz` must be factorable by 2, 3, 5 and at least 2. Note that `ny` is the number of Chebyshev polynomials and thus is equal to $N_y + 1$ used in section 2 above.

Dealiazing, i.e. padding to remove aliazing errors, can be switched on (=1) or off (=0) independently for each direction by the flags `nfxd`, `nfyd` and `nfzd`. If

dealiazing in the respective direction is used `nx`, `ny-1` must be divisible by 4, and `nz` must be divisible by 2. Z-symmetry can be used to reduce computation time and storage by setting `nfzsym`=1. If this is done `nz` must be divisible by 4, and if used simultaneously with dealiazing in the z-direction `nz` must be divisible by 8.

There is an option to run 2 1/2 dimensional simulations, i.e., simulations of flow in a two dimensional geometry with all three velocity components non-zero, which is sometimes called the infinite swept flow. (Two dimensional flow is a special case of this.) In this case set `nz`=1 `nfzsym`=0 and `nfzd`=0. (In this case the limitations on `nz` given above do not apply.)

Normally (`nfc`=1) all the storage resides in primary memory but it is possible to put the main three dimensional arrays in the external file `ur` by setting `nfc`=0. To achieve maximum performance, especially for external main storage, the parameters `mby` and `mbz` can be changed from the default value =1, see section 4.4 below. Note that `nz` must be divisible by `mbz`. The program can be coarse grain parallelized, in which case the parameter `nproc` should be changed from the default value one to the number of available processors. This is also discussed in section 4.4. To allow for simultaneous calculation of velocities and pressure, the parameter `pressure` should be set to 1. All other parameters in the **par.f** file are computed and should not be changed manually. Note that most subroutines except those in the libraries **dclib**, **VECFFT** and **plot1** must be recompiled after changing **par.f**.

The codes are written in single precision, i.e. with REAL and COMPLEX declaration. However, in most cases there is a need to run the code in double precision, i.e., with at least 10-12 digit precision. For this purpose the supplied makefile convert the programs to double precision. Note that for the programs to work with the libraries and together with binary files all routines must be compiled with the same precision. The makefile automatically compile the libraries with the same precision as the program. For the double precision, you can also use the compiler option by specifying the default size of variables as DOUBLE PRECISION like "-r8". This option varies from one machine to another. In case of work stations, the compiler option for the double precision is used. See the Makefile for more informations.

However, to change precision (i.e., compiling the programs as double precision where they have previously been compiled as single or vice versa) it is necessary to delete all object files before recompiling. This is not handled automatically by the makefiles.

The same makefile named "Makefile" can be used in most machines including Crays, IBM, SGI, SUN, DEC and HP. You must have "cpp" in path and may need to change preprocessor option because it varies from machine to machine. The C language preprocessor, cpp performs the preprocessing directives in some

programs like ctim.f. It is useful to handle system dependent functions in one file.

4.2. *Generation of initial velocity fields with* **bls**

An initial velocity field consists of a header and an array with the three components of velocity in Fourier space fulfilling the equation of continuity. The format of the file is described in section 5.3. The routine **bls** may be used to generate an initial velocity field, consisting of a basic laminar flow, a localized disturbance, waves and a random noise. The different disturbances can be switched off to allow zero to three disturbances to be inserted.

The initial velocity field file has the same format as files generated by subsequent execution of the **bla** program so that it is possible to feed the initial velocity field to the postprocessing directly for examination.

To compute a velocity field a velocity profile file must first be generated. The subroutine **fskch** finds velocity profiles from the Blasius/Falkner-Skan/Falkner-Skan-Cooke family. These are similarity boundary layer profiles derived from the laminar boundary layer flow equations for flow over a flat plate, wedge and infinite swept wedge. **bls** is generating a temporal/spatial or parallel/non-parallel velocity field depending on flow type parameter **fltype**.

bls is intended for batch execution and has no interactive input. The input comes from the file **bls.i**. The format of this file is given in section 5.1. All input is non-dimensionalized with the displacement thickness and free-stream velocity at the inflow boundary ($x = 0$) at $t = 0$.

4.3. *Generation of non-similarity base flows*

In case the streamwise free-stream velocity is not a power of the downstream distance, the boundary layer equations do not have a self similar solution. To generate a base flow for this situation we can first use **bls** to generate a similarity flow field (without disturbances) which is a good approximation to the sought flow around the inflow boundary. I.e., a flow such that boundary layer thickness and acceleration are correct around the inflow boundary. Then this flow field can be advanced in time with **bla** to find a steady state using a streamwise free-stream velocity given in tabular form as a function of the downstream distance (see section 5.2 and 5.10). The generated steady flow field can be input to **bls** and disturbances superimposed. The same flow field can be used to specify the baseflow to **bla** for subsequent simulations.

4.4. *Execution of* **bla**

The program is intended to be used in batch mode and so has no interactive input. The main configuration is done at compile time through changes to the

file **par.f** (see section 4.1) and at runtime by **bla.i** (see section 5.2). An initial velocity field, which can be produced by the program **bls**, see above, is needed to start execution.

4.4.1. *Storage requirements*

The core size depends on the compiled size of the code, the resolution of the simulation, and whether dealiazing in the y-direction is used, the tuning parameters `mby`, `mbz` and `nproc` and if the three dimensional storage is in the core.

The two dimensional storage for step 2 is `7*nx*nz*mby*nproc` words; multiply by a factor 1.5 each for dealiazing in the x and z-directions, by 0.5 for z-symmetry and by 8/7 if the pressure solver is activated. For step 3 storage is `19.5*nx*ny*mbz*nproc` words; multiply by 1.5 for dealiazing in the y-direction and 8/7 for pressure solver. The storage for step 2 and step 3 overlaps so that the total two-dimensional storage is equal to the maximum of the requirement for step 2 and step 3.

The three dimensional storage is `7*nx*ny*nz` words, multiply by a factor of 1.5 for dealiazing in the y-direction, by 0.5 for z-symmetry. This storage can be kept out of the core by setting `nfc`=0.

4.4.2. *Tuning*

The code itself has been written for maximum speed on a vectorizing computer using a highly optimizing compiler. To achieve highest possible performance the main storage should preferably be kept in the core. If this is is not possible the performance in terms of wall time will degrade due to waiting for I/O, but the CPU time will only increase in the order of 10%.

For tuning of the program to a given installation two parameters `mby` and `mbz` can be set in **par.f**. This has the greatest impact on performance if the storage of the main data is out of core. For large in-core simulations `mby`=`mbz`=1 will generally give good performance. Note that `nz` must be divisible by `mbz`.

If the three dimensional storage is in the core the value of `mby` and `mbz` affects only the vector lengths. The basic vector length is `nxp/2*(mby-1)+nx/2` in most of step 2 (where `nxp` is equal to `nx` without x-dealiazing and `nx*3/2` with x-dealiazing, and `nz*mby` in the x-transform, multiply the latter by 1.5 for z-dealiazing and multiply by 0.5 and add 1 for z-symmetry. The vector length in step 3 is `nx/2*mbz`. If these values are lower than what is needed to get a good performance, `mby` and `mbz` can be increased.

If the three dimensional storage is out of core it is important to keep the record length, `nx*mby*mbz*npreal` bytes (where `npreal` is the number of bytes used to store a real number, in communication with the main storage file as large as

possible. Since increasing `mby` and `mbz` increases the amount of internal storage, this is preferably done by balancing the amount of storage needed for step 2 and step 3, cf. above. A suggestion is to put `mby`=`mbz`=2 and see if this gives an acceptable performance in terms of wall time/CPU time. If not, they can be increased to see if this improves the situation. Note finally that nothing can be done to the finite bandwidth of the transfer between disk and processor, the program will do about 4 flops for every byte transferred between disk and processor (8 when using 4-byte reals), so it is quite likely that the program will spend a large portion of the time waiting for the disk.

The program is prepared to be coarse-grain parallelized. Step 2 and step 3 can each be divided on as many processors as there are boxes to process; typically this is no limitation. There are directives for several compilers inserted before the loops 2 and 3, these may have to be replaced for compilers not previously used. To achieve parallelization `nproc` in **par.f** should be set to the number of processors to be used. Then all subroutines have to be compiled as recursive, i.e. with dynamic local storage. In addition a parallelizing option has to be added to the compile statement for the main program. The code has been run in parallel mode on the Alliant FX-80 and FX-2800, the SGI Powerstation, Challenge and Power Challenge, the CRAY-2, J90 and C90. The typical speed-up is 3.5-3.8 for four processors.

A slightly different version of the code has been implemented on various computers with distributed memory, such as IBM SP2 and CRAY T3E. The communications between the processors are handled with the Message-Passing Interface (MPI). The efficiency has been tested and is reported in Alvelius & Skote (2000).

4.5. *Post processing*

4.5.1. *Post processing velocity files with* **pre** *and* **ritpre**

The program **pre** generates a pressure field from a velocity field. The pressure can be examined with the program **ritpre** in the same way a velocity field is post processed with **rit**.

4.5.2. *Post processing velocity files with* **rit**

The program **rit** generates various graphs from a velocity field file. The graphs can be generated in either Tektronix 4014 format or Postscript. There is also a possibility to produce black and white portable pixel maps (ppm). When executed, **rit** prompts for an input file name. The file is read and the program offers a choice of various types of graphs. It is mainly intended for interactive execution and should be self explanatory.

It is possible to use **rit** in a batch environment by compiling it into a input program. This is run interactively to produce a file **ritin**, which is subsequently read by the batch code to produce the desired plots. Note that if plots in batch mode are produced 'to the screen' the resulting Tektronix graphic characters will be written to the log file. To compile a batch program, set `imode` to 2 in the **rit.f** file and compile a second time with `imode`=3 to get an input program. To get an interactive program `imode` should be left at 1.

4.5.3. *Post processing velocity files with* **cmp**

The program **cmp** is used for subtracting and adding different velocity fields. This is useful when comparing velocity fields.

4.5.4. *Post processing plane files with* **rps**

Planes saved during a simulation can be examined with the program **rps**.

4.5.5. *Post processing velocity files with* **fou**

When a number of velocity fields has been saved during a simulation, the program **fou** can be used to make Fourier transforms in both time and space.

4.5.6. *Postprocessing amplitude files with* **pamp1**, **pamp2**, **pampw**, **pampw**, **pampw2** *and* **pext1**

The programs **pamp1** and **pamp2** can be used to produce plots of the time history of various amplitudes from the amplitude files by written **bla**. **pamp1** works on one file and **pamp2** can plot one quantity from multiple files. **pext1** makes plots of time histories of extremum values (i.e. min and max values) of velocities and vorticities and the location of extrema. **pampw** and **pampw2** similarly plot amplitudes of wave components (streamwise-spanwise Fourier mode) from one or multiple wave-amplitude files. The programs are intended to be self explanatory and prompt for input file names. Since the amplitude files are formatted and normally relatively small, no batch versions of these programs are available. The files contain no headers so that files from sequential runs of one flow case can be concatenated and then plotted to show the complete evolution of the amplitudes.

4.5.7. *Postprocessing xy-statistics files with* **pxyst**

To get good statistics of space developing flows with one homogeneous direction (spanwise), the data needs to be averaged in time. The plotting of time and spanwise averaged data saved to file is performed by **pxyst**. Note that these files have headers, and thus cannot be concatenated together. The statistics in

different files can be added together by the program **addxyst**. The format of the statistics files is given in section 5.9 below.

pxyst generates plots both of the raw statistical data and of a number of derived quantities. It is also possible to generate various special plots of the mean flow, such as boundary layer thicknesses and skin friction.

There is an initial option to filter data, which applies to the raw data, before computing other quantities. There is also an option to filter data prior to producing plots, the filter is then applied to the derived quantity. The results of the two filtering processes may differ. In both cases the filter is applied in the streamwise direction.

5. File formats

These are the input/output files used by the programs. For the format of the external main storage file see section 3.2.2 above.

5.1. **bls.i** *file*

bls.i is formatted and sequential. Comments can be put after data on lines not containing character input. All input is non-dimensionalized with the displacement thickness at $x = 0$, $t = 0$ and the free-stream velocity at $x = 0$, $t = 0$. For more explanations see section 4.2. Contents line by line :

1. `namnin` Optional input velocity field file name; character*32.

As an option the base flow can instead be given in the form of an input velocity field file.

2. `namnut` Output velocity file name; character*32.

3. `re` The Reynolds number (based on the units above); real.

4. `xlb` The length of the computational box; real.

5. `h2` The height of the computational box; real.

6. `zlb` The width of the computational box; real.

The dimension of the simulation box in all three dimensions must be given. The streamwise extent of the box must for spatially developing flows include the length of the fringe region, which is typically set to 30-100 displacement thicknesses. The vertical extent of the box must include the whole boundary layer. Depending on the choice of free-stream boundary condition, the box may include only the boundary layer or a few times more. The sufficiency of the box height may be investigated through numerical experiments.

7. `fltype` Type of flow (-2 temporal Falkner-Skan-Cooke, -1 temporal Falkner-Skan, 3 temporal Blasius BL, 6 spatial Blasius BL, 7 spatial Falkner-Skan, 8 spatial Falkner-Skan-Cooke, 9 spatial parallel Blasius/Falkner-Skan/Falkner-Skan-Cooke; integer.

8. If `fltype` = -1 or $\geq$ 7: `rlam` The acceleration exponent of the velocity in the free-stream; real.

9. If `fltype` = -2 or $\geq$ 8: `spanv` The spanwise free-stream velocity; real.

10. If `fltype` $\geq$ 6: `bstart` The x-value of the start of the blending of the base flow; real.

11. If `fltype` $\geq$ 6: `bslope` The length of base flow blending region; real.

The base flow can either be parallel or space developing. The parallel base flow is for the present version only of Blasius type and is selected by setting `fltype`=3. The space developing base flow can be either Blasius (`fltype`=6), Falkner-Skan (`fltype`=7), or Falkner-Skan-Cooke (`fltype`=8). For the two latter the acceleration exponent `rlam` for the streamwise free-stream velocity must be given (i.e. m in $U = Cx^m$). For Falkner-Skan-Cooke (swept wedge) flow the spanwise velocity in the free-stream must be specified. Note that the spanwise direction is parallel to the leading edge of the wedge for this case, and that the spanwise free-stream velocity is constant. For spatially developing flows the base flow from the upstream and the downstream end are blended in the fringe region. The start and blending length must be specified. Typically the start is given as a negative number i.e., the distance upstream of the inflow boundary where the blend starts is given. (see section 2.4)

12. `ushift` The Galilei shift velocity, =0 for no shift; real.

13. `locdi` Flag to generate a localized disturbance; logical.

13.a If `locdi` is true: `ditype` The type of disturbance , only useful values 1 to 3; integer.

13.b If `locdi` is true: `amp` The amplitude of a localized disturbance; real.

13.c If `locdi` is true: `theta` The rotation angle of the localized disturbance in radians; real.

13.d If `locdi` is true: `xscale` The streamwise scale of the disturbance; real.

13.e If `locdi` is true: `xloc0` Origin of the disturbance in x-direction; real.

13.f If `locdi` is true: `yscale` The wall normal scale of the disturbance; real.

13.g If `locdi` is true: `zscale` The spanwise scale of the disturbance; real.

13.h If `locdi` is true: `ipoly` The wall normal distribution of the disturbance, only useful values 1 to 4; integer.

The `ditype` determines the type of disturbance. See **bls.i** for more information. The example below is for `ditype` set to 1.

The localized disturbance is governed by the amplitude, the rotation angle, the length and spanwise scale. The rotation angle is the angle by which the spanwise symmetric disturbance is rotated about the y-axis. The x-scale and the z-scale of the disturbance are given to be applied to the disturbance before rotation. The form of the disturbance is in a coordinate system aligned with disturbance:

$$\begin{aligned} u' &= 0 \\ v &= -\frac{\partial \psi}{\partial z} \\ w' &= -\frac{\partial \psi}{\partial y} \\ \psi &= amp \frac{x'}{x_{scale}} \frac{z'}{z_{scale}} p(\frac{y}{y_{scale}}) e^{-(\frac{x'}{x_{scale}})^2 - (\frac{z'}{z_{scale}})^2} \end{aligned} \tag{99}$$

where $p(s)$ is determined by `ipoly`, see **bls.i**. The relation between the disturbance aligned velocities and coordinates (with $'$) and the computational box aligned ones is :

$$x = x' \cos(theta) + z' \sin(theta) \tag{100}$$

$$z = -x' \sin(theta) + z' \cos(theta) \tag{101}$$

$$u = w' \sin(theta) \tag{102}$$

$$w = w' \cos(theta) \tag{103}$$

14. `waves` Flag to generate a pair of oblique waves; logical.

14.a If `waves` is true: `energy` Energy density of the waves; real.

14.b If `waves` is true: `ystart` The lowest y-value of non-zero wave amplitude; real.

14.c If `waves` is true: `yend` The largest y-value of non-zero wave amplitude; real.

14.d If `waves` is true: `yrise` The switch distance from zero to max wave amplitude; real.

14.e If `waves` is true: `yfall` The highest y-value of non-zero wave amplitude; real.

14.f If `waves` is true: `walfa` Streamwise wave number of the waves; real.

14.g If `waves` is true: `wbeta` Spanwise wave number of the waves; real.

15. `os` eigen modes flag, `.true.` for use of tabulated eigen modes; logical.

16. `noise` noise flag, `.true.` for noise; logical.

16.a If `noise` is true: `ed` The mean energy density of the noise; real.

16.b If `noise` is true: `nxn` The maximum streamwise wavenumber of the noise, should be $\leq$ `nx`/2; integer.

16.c If `noise` is true: `nyn` The number of vertical Stokes modes in the noise, should be even, $<$ `ny*2/3`; integer.

16.d If `noise` is true: `nzn` The maximum spanwise wavenumber of the noise, should be odd, $<$ `nz`; integer.

16.e If `noise` is true: `seed` A random number seed in the range -700000 to -1; integer.

The noise is in the form of Stokes modes, i.e., eigenmodes of the flow operator without the convective term. These fulfill the equation of continuity and the boundary condition of vanishing velocity at the lower and upper boundaries. Although the actual boundary condition may allow a non-zero amplitude at the free-stream boundary the restriction of zero amplitude for the noise doesn't have a large impact in practise.

The noise can be switched on by a flag in the input file. If noise is used the mean energy density must be given along with the number of wave numbers to be randomized for each direction. In the wall normal direction the number of Stokes modes to be randomized is given. The same noise will be generated for the same setting of this seed, if the physical size of the simulation box is unchanged. In particular the resolution can be changed without affecting the noise, as long as the number of grid points is sufficient to resolve the noise modes. This is useful for convergence studies.

5.2. **bla.i** *file*

bla.i is formatted and sequential. Comments can be put after data on lines not containing character input. For more explanations see section 4.4. Contents line by line :

1. `namnin` Input velocity file name; character*32.

2. `namnut` Output velocity file name; character*32.

3. If `pressure` in **par.f** is 1: `namnutp` Output pressure file name; character*32.

4. `tmax` The final time to which to simulate; real.

5. `maxit` The maximum number of iterations to simulate; integer.

6. `cpumax` The maximum CPU time in seconds; real.

The input and output file names and the final time `tmax` determine the scope of the simulation, in addition setting the maximum number of iterations puts a limit on the number of iterations to be taken through the main time step loop. The latter parameter is useful with variable time stepping in a batch environment to ensure that the execution terminates before running out of execution time. If the maximum number of iterations is used before the final time is reached the execution will terminate normally by saving the present velocity field to the output velocity file. Note that for RK3 a time step consists of three or four iterations. The execution will only stop after completing an integer number of physical time steps. If adaptive time stepping is used the program will adjust the final four time steps so that it reaches exactly the final time. You can also control maximum execution time in CRAY systems by giving the maximum CPU time for batch job so that it terminates by `cpumax`. You just give a very big number if you do not need to control maximum execution time.

7. `dt` The time step length; real.

`dt` is the length of the time step, if it is set ≤ 0 the adaptive time stepping is used. The time step is regulated to keep the CFL number close to `cflmax`, which is set to $0.9\sqrt{3}$ for the three stage Runge-Kutta and $0.9\sqrt{8}$ for the four stage Runge-Kutta. When using a fringe region the time step is also limited by the numerical stability for the damping term, this is $1.75/$`fmax` for the three stage RK and $1.96/$`fmax` for the four stage RK (`fmax` is the max strength of the fringe region, see below). If `dt` is set < 0 then $-$`dt` is used as an additional limit on the variable time step.

8. `nst` The number of stages in the time discretization; integer (3 three stage Runge-Kutta, 4 four stage Runge-Kutta).

`nst` selects between the different formulas for the explicit time discretization. The 4 stage Runge-Kutta method is about 20% more efficient than the 3 stage version.

9. `xl` The new box length. If lower than the old length, the old value will be used; real.

10. `varsiz` Flag to allow read of a file of different size than the code is compiled for; logical.

If `varsiz` is set true the program may start from an input field of a different resolution than the program is compiled for. The spectral coefficients are

ibc	BC at free-stream boundary
0	$u = v = w = 0$
1	$Du = Dv = Dw = 0$
2	$D^2u = D^2v = D^2w = 0$
3	$D^3u = D^3v = D^3w = 0$
10	$Du + ku = Dv + kv = Dw + kw = 0$
11	$D^2u + kDu = D^2v + kDv = D^2w + kDw = 0$
12	$D^3u + kD^2u = D^3v + kDv = D^3w + kD^2w = 0$
20	$Dv + kv = D^2v + kDv = \omega = 0$
100	$u = U, v = V, w = W$
101	$Du = DU, Dv = DV, Dw = DW$
110	$Du + ku = DU + kU, Dv + kv = DV + kV, Dw + kw = DW + kW$
120	$Dv + kv = DV + kV, D^2v + kDv = D^2V + kDV, \omega = 0$
130	$u = U, Du = DU, w = W$
140	$u = U, v = dDV, w = W$
150	$u = U, Du - vx = 0, Dw = 0$

TABLE 2. Free-stream boundary conditions. u, v, w are the solution velocities. U, V, W are the base flow velocities. D is the velocity derivative normal to the boundary, k is the modulus of the horizontal wavenumber ($k^2 = \alpha^2 + \beta^2$).

padded with zeroes or truncated to achieve a spectrally accurate interpolation. However, the resolution cannot be reduced in the normal direction as the truncated field in general will not fulfill the equation of continuity and the boundary conditions.

11. `rot` The rotation rate, 0. for no rotation ; real.

`rot` is the angular velocity of the coordinate frame around the z-axis. For non-rotating flows it should be set to zero.

12. `ibc` The boundary condition number; integer.

`ibc` is the number of the free-stream boundary condition. The implemented boundary conditions are given in table 2. See also section 2.2 above. A number of these boundary conditions makes the numerical scheme unstable. Among the stable boundary conditions , the most used are number 101 and 110.

13. `cim` Flag to use chebyshev integration method. If false the tau method is used; logical.

14. If `cim` is true: `icorr` Flag to use integration correction; logical.

`icorr` is a flag to use integration correction. The combination of using integration correction and boundary conditions other than of Dirichlet type may lead to numerical instability. The flag is normally set false.

15. `gall` Flag to compute and use a Galilei transformation to increase max stable time step; logical.

16. `spat` Flag to perform spatial simulation; logical.

`spat` turns on spatial simulations, if it is set false the program performs a temporal simulation. For spatial simulations a number of parameters specifying the fringe region must be given, see section 2.4 above.

17. If `spat` is true: `tabfre` Flag to use a tabulated free-stream velocity; logical.

To use a tabulated free-stream velocity the flag `tabfre` is set true. The format of the free-stream velocity file is given in section 5.10 below.

18. If `spat` and `tabfre` are true: `namfre` Name of file containing free-stream velocity table; character*32.

19. If `spat` is true: `rbfl` Flag to use a 3-d flow field as a base flow; logical.

To use a 3-d flow file to define the base flow the flag `rbfl` is set true. The format of the 3-d flow file is given in section 5.3 below.

20. If `spat` and `rbfl` are true: `nambfl` Name of file containing a 3-d base flow; character*32.

21. If `spat` is true : `fmax` Maximum strength of the fringe region; real.

22. If `spat` is true : `fstart` x-position of the start of the fringe region; real.

23. If `spat` is false : `fend` x-position of the end of the fringe region; real.

24. If `spat` is true : `frise` The distance from the start of the fringe region to the first point of maximum damping; real.

25. If `spat` is true : `ffall` The distance from the last point of maximum damping to the end of the fringe region; real.

26. If `spat` is true : `ampob` The amplitude of oblique waves forced in the fringe; real.

A pair of oblique waves can be generated in the fringe region by setting `ampfw` non-zero. The format of the waveform file **wave.d** is given in section 5.11.

27. If `spat` is true : `amp2d` The amplitude of two dimensional T-S wave forced in the fringe; real.

28. If `spat` is false : `cdev` The reference speed for the parallel boundary layer growth; real.

For temporal simulations `cdev` must be set to the reference speed of the boundary layer growth, see section 2.3 above.

29. `loctyp` to generate a localized volume force disturbance; integer.

`loctyp` can take values from 1 to 5. Various disturbances can be created. See **locf** for more information. The different values of `loctyp` each require a distinct number of parameters in the file **bla.i**, see **rparambl** for more information. As an example, a localized volume force disturbance to generate wave packets is created by setting `loctyp` to 1. The following parameters (28.a-i) are required if `loctyp` is 1. Different parameters are needed when `loctyp` is 2, 3, 4 or 5. This is explained in **rparambl** and **locf**.

29.a If `loctyp` is 1 : `ampx` Max amplitude of the localized volume force disturbance in x-direction; real.

29.b If `loctyp` is 1 : `ampy` Max amplitude of the localized volume force disturbance in y-direction; real.

29.c If `loctyp` is 1 : `ampz` Max amplitude of the localized volume force disturbance in z-direction; real.

29.d If `loctyp` is 1 : `xscale` Length scale of the localized volume force disturbance in x-direction; real.

29.e If `loctyp` is 1 : `xloc0` Origin of the localized volume force disturbance in x-direction; real.

29.f If `loctyp` is 1 : `yscale` Length scale of the localized volume force disturbance in y-direction; real.

29.g If `loctyp` is 1 : `zscale` Length scale of the localized volume force disturbance in z-direction; real.

29.h If `loctyp` is 1 and `zscale` < 0 : `lskew` The obliqueness of waves of the localized volume force disturbance; real.

29.i If `loctyp` is 1 : `tscale` Time scale of the localized volume force disturbance; real.

If `loctyp` is 1, the form of the localized disturbance is:

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} = \begin{pmatrix} amp_x \\ amp_y \\ amp_z \end{pmatrix} e^{-(y/y_{scale})^2} g(x,z) f(t), \tag{104}$$

where

$$\begin{aligned} z_{scale} > 0 \quad g(x,z) &= e^{-[(x-x_{loc0})/x_{scale}]^2 - (z/z_{scale})^2} \qquad (105) \\ z_{scale} < 0 \quad g(x,z) &= \cos(2\pi(z - x_{lskew})/z_{scale}) e^{-}[(x - x_{loc0})/x_{scale}]^2, \end{aligned}$$

and

$$\begin{aligned} t_{scale} > 0 \quad f(t) &= e^{-(t/t_{scale})^2} \\ t_{scale} < 0 \quad f(t) &= S(-t/t_{scale})) \\ t_{scale} = 0 \quad f(t) &= 1, \end{aligned} \tag{106}$$

and

$$S(x) = \begin{cases} 0 & x \le 0 \\ 1/[1+\exp(\frac{1}{x-1}+\frac{1}{x})] & 0 < x < 1 \\ 1 & x \ge 1 \end{cases} . \tag{107}$$

30. `tripf` Flag to generate a random "sandpaper" volume force trip strip; logical.

30.a If `tripf` is true : `tamps` Max stationary amplitude of the trip; real.

30.b If `tripf` is true : `tampt` Max time varying amplitude of the trip; real.

30.c If `tripf` is true : `txsc` x length scale of the trip; real.

30.d If `tripf` is true : `tx0` x origin of the trip; real.

30.e If `tripf` is true : `tysc` y length scale of the trip; real.

30.f If `tripf` is true : `nzt` Number of z Fourier modes in the trip; integer.

30.g If `tripf` is true : `tdt` Time interval between change of the time dependent part of the trip; real.

30.h If `tripf` is true : `seed` Negative number in the range -700000 to -1 to initialize the random number generator for the trip ; integer.

`tripf` is a flag to enable forcing of a volume force trip strip at the wall running in the spanwise direction. The trip can be used to generate turbulence or at lower amplitude levels to test the stability of a boundary layer or flow structure. The trip has a steady amplitude `tamps`, and a time dependent amplitude `tampt` which allow both steady and time varying trips to be generated. The volume force has one continuous time derivative and is independent of the time discretization. The random numbers are generated such that if the random number `seed` and other trip parameters are unchanged, the same trip forces are generated. This is true even if the simulation is split into two or more runs. For every run beyond the first the random number generator is run forward to the correct state. The form of the volume force, which is directed normal to the wall, is as follows :

$$F_2 = \exp[((x - t_{x0})/t_{xsc})^2 - (y/t_{ysc})^2]f(z,t), \tag{108}$$

where

$$f(z,t) = t_{amps}g(z) + t_{ampt}[(1 - b(t))h^i(z) + b(t)h^{i+1}(z)], \tag{109}$$

and

$$\begin{aligned} i &= \mathrm{int}(t/t_{dt}), \\ b(t) &= 3p^2 - 2p^3, \\ p &= t/t_{dt} - i. \end{aligned} \tag{110}$$

$g(z)$ and $h_i(z)$ are Fourier series of unity amplitude with `nzt` random coefficients.

31. `wbci` Boundary conditions at wall; integer.

`wbci` can be set to 0, 1, 2 or 3. If `wbci` is not equal to zero, additional parameters must be provided. See **rparambl** and **cwallbc**. The example below is for `wbci` set to 1.

31.a If `cwallbc` is 1 : `amp` Max amplitude of the localized blowing/suction; real.

31.b If `cwallbc` is 1 : `damp` Damp amplitude. No effect if less than one; real.

31.c If `cwallbc` is 1 : `xstart` Start position of disturbance; real.

31.d If `cwallbc` is 1 : `xend` End position of disturbance; real.

31.e If `cwallbc` is 1 : `xrise` Rise length of disturbance; real.

31.f If `cwallbc` is 1 : `xfall` Fall length of disturbance; real.

31.g If `cwallbc` is 1 : `zbet` Spanwise variation; real.

31.h If `cwallbc` is 1 : `tomeg` Time variation; real.

The blowing and suction at the wall is implemented in **cwallbc**. The form of the boundary condition below is for `wbci` set to 1.

$$v|_{y=0} = amp \cdot f(x) \cdot cos(zbet \cdot z) \cdot sin(tomeg \cdot t), \tag{111}$$

where

$$f(x) = S\left(\frac{x - x_{start}}{x_{rise}}\right) - S\left(\frac{x - x_{end}}{x_{fall} + 1}\right), \tag{112}$$

and $S(x)$ is given by equation (107).

32. `icfl` Number of time iterations between calculation of the CFL number; integer.

`icfl` is the calculation interval for the CFL number. If the CFL number is computed each iteration this adds a few percent to the execution time, but since it is used to regulate the time step it should not be computed too sparsely, preferably every complete time step, i.e. `icfl` = `nst`.

33. `iamp` Number of time iterations between calculation of rms amplitudes; integer.

33.a If `iamp` > 0 : `namamp` Output file for rms amplitudes; character*32.

`iamp` is the interval for evaluation of the amplitude. As for the CFL number continuous calculation of the amplitude costs a number of percent in execution speed. If `iamp`=0 no amplitudes will be calculated and no amplitude file will be written. To get the correct time accuracy `iamp` should be an integer multiple of `nst`.

34. `longli` Flag to generate amplitude for each horizontal plane (y-value). Applies both to rms amplitudes (items 46-47) and wave component amplitudes (items 58-60).

`longli` is set true the program will produce y-dependent statistics and write these to the amplitude files, both for the global statistics and statistics by wavenumber. The statistics files can become quite long if the flag is set true.

35. `iext` Number of time iterations between calculation of extremum amplitudes; integer.

`iext` is the interval for evaluation of the extremum values and their coordinates. This evaluation is somewhat more time consuming than that for the amplitudes. If `iext`=0 no extremum amplitudes will be calculated. To get the correct time accuracy `iext` should be an integer multiple of `nst`.

35.a If `iext` > 0 : `namext` Output file for extremum amplitudes; character*32.

36. `ixys` Number of time iterations between calculation of xy-statistics; integer.

`ixys` is the interval for evaluation of xy-statistics, used by **pxyst**. The statistics generated and the output file format are described in section 5.9. The file is written to every `ixyss` iterations, overwriting older data. To get the correct time accuracy `ixys` should be an integer multiple of `nst`.

36.a If `ixys` > 0 : `namxys` Output file for xy-statistics; character*32.

36.b If `ixys` > 0 : `ixyss` Number of time iterations between saving of xy-statistics data to file; integer.

36.c If `ixys` > 0 : `txys` Time to start accumulation of xy-statistics; real.

37. `msave` The number of complete intermediate velocity fields to be saved. If non-zero, items a and b are repeated for each file; integer.

37.a If `msave`> 0 : `tsave` The time for which to save an intermediate field; real.

37.b If `msave`> 0 :`nmsave` The name of the intermediate velocity file; character*32.

`msave` is the number of intermediate velocity fields to be saved, maximum 20. If higher than zero the times and names of the files to be saved must be given. If the time stepping is adaptive the program automatically adjusts the time step to reach exactly the desired times. For fixed time step the save is done at the nearest time.

38. `mwave` The number of wavenumbers to save amplitudes for. If non-zero, item b is repeated for each wavenumber; integer.

38.a If `mwave`> 0: The name of the wave amplitude file; character*32.

38.b If `mwave`> 0: `kx`,`kz` The streamwise wavenumber as multiples of the fundamental $2\pi/x_L$, the spanwise wavenumber as multiple of the fundamental $2\pi/z_L$; both integers.

`mwave` sets the number of specific wavenumbers to calculate amplitudes for. For each wave, the x and z wavenumbers must be specified as integers to be multiplied by $2\pi/x_L$ and $2\pi/z_L$ respectively. The wavenumbers are counted in the physical way for positive and negative `kz` and `kx` zero and up, not in the way of the internal storage. The wave amplitudes are calculated for each of the six velocities and vorticities at intervals set by the `iamp` value.

39. `npl` The number of planes to be continuously saved during the simulation. If non-zero, items b through e are repeated for each plane; integer.

39.a If `npl` > 0: `ipl` The saving interval for planes in number of iteration; integer.

39.b If `npl` > 0: `tpl(i,1)` The type of plane to be saved. 1 for xy,2 for xz; integer.

39.c If `npl` > 0: `tpl(i,2)` The variable to be saved, i.e. 1 for u, 2 for v, 3 for w; integer.

39.d If `npl` > 0: `cpl` The coordinate for which to save the plane; real.

39.e If `npl` > 0: `nampl` The name of the file in which to save the planes; character*32.

`npl` is the number of 2d planes to be saved every `ipl` iterations during the simulation. To get the correct time accuracy `ipl` should be an integer multiple of `nst`. It is these files which are used by **rps** for plotting, the format is described in section 5.8 below.

5.3. *Velocity file*

Format of a 3-d uncompressed velocity file. The format is used for any 3-d input or output from **bls** and **bla**. The file is unformatted, sequential.

Record 1: Reynolds number; real, `.false.` (this is to be backward compatible with channel flow files); logical, x_L; real, z_L; real, the time for this field; real, the length by which the box has been shifted to the right since time zero; real.

Record 2: Number of spectral modes in the x-direction; integer, the number of points in the physical y-direction; integer, the number of spectral modes in the z-direction reduced for symmetry; integer, 0/1 no z-symmetry/z-symmetry; integer.

Record 3: Flow type `fltype`; integer, displacement thickness expressed in half box heights `dstar`; real.

Record 4: If `fltype` ≥ 4 : start of blending region `bstart`, end of blending region `bslope`, if `fltype` ≥ 7: acceleration exponent of streamwise free-stream velocity `rlam`, spanwise free-stream velocity `spanv`. For other values of `fltype` this record is omitted.

Record 5: The u, v, w-velocities in Fourier x, z and physical y space. One record contains `nx`/2 complex coefficients in normal Fortran format. The records are stored in `y`, `z`, `i` order with `y` varying the fastest and `i` the slowest. The number of points in the y-direction is `nyp` and the number in the z-direction `nzc`. Total number of records `nyp*nzc*3`.

5.4. *Pressure file*

Format of a 3-d uncompressed pressure file. The format is the same as for the velocity file, except the last record which contains only the pressure.

5.5. *Amplitude file*

Formatted, sequential. The rms-levels are an average over the physical box. For each time three records are saved:

1. Time; real, u_{rms}; real, v_{rms}; real, w_{rms}; real.

2. χ_{rms}; real, ω_{rms}; real, ϑ_{rms}; real,ω^2/k^2; real.

3. $DUuv$; real, energy for wavenumber zero; real, h^+, i.e. the box half-height in wall units; real.

if `longli` is `.true.` then for each time the above is followed by statistics by y-plane in descending y-coordinate order as follows :

4. mean squared streamwise velocity without Blasius base flow; real, mean squared normal velocity ; real, mean squared spanwise velocity ; real, mean squared streamwise vorticity ; real, mean squared normal vorticity ; real, mean squared spanwise vorticity without Blasius base flow ; real, mean squared vorticity squared over wavenumber square average, no (0,0); real, Reynolds stress

average; real, mean streamwise disturbance velocity squared; real, mean spanwise disturbance velocity squared; real.

5.6. *Wave amplitude file*

Formatted, sequential. The data in this file is in internal scaling. For each time are saved:

1. Time; real, number of waves saved; integer, number of points in the y-direction; integer, Reynolds number; real, fundamental wavenumber in the x-direction; real, fundamental wavenumber in the z-direction; real, flag `longli`.

2. The wavenumber α as multiples of the fundamental $2\pi/x_L$; integer, the wavenumber β as multiple of the fundamental $2\pi/z_L$; integer, u_{rms}; real, v_{rms}; real, w_{rms}; real, ω_{rms}; real.

Item 2 is repeated for each wave.

if `longli` is `.true.` then for each time the above is followed by statistics by y-plane in descending y-coordinate order as follows :

3. if the wavenumber is zero : $\hat{u}$ for each y-plane (with the imaginary part zero), otherwise $\hat{v}$ for each y-plane; complex.

4. if the wavenumber is zero : $\hat{w}$ for each y-plane (with the imaginary part zero), otherwise $\hat{\omega}$ for each y-plane; complex.

Item 3 and 4 are repeated for each wave.

5.7. *Extremum file*

Formatted, sequential. For each time are saved:

1. Time; real.

2. Min $u - U_{laminar}$; real, x-coordinate for this minimum; real.

3. y-coordinate; real, z-coordinate; real.

4. and 5. same for min v

6. and 7. same for min w

8. and 9. same for min χ

10. and 11. same for min ω

12. and 13. same for min ϑ

14. and 15. same for min $\vartheta - \vartheta_{laminar}$

16. through 29. same as 2. through 15. but for maximum

5.8. *Plane velocity file*

Unformatted, sequential.

Record 1: Reynolds number; real, `.false.` (this is to be backward compatible with channel flow files); logical, x_L; real, z_L; real, the time for this field; real, the length by which the boxed has been shifted to the right since time zero; real.

Record 2: Number of spectral modes in the x-direction; integer, the number of points in the physical y-direction; integer, the number of spectral modes in the z-direction reduced for symmetry; integer, 0/1 no z-symmetry/z-symmetry; integer.

Record 3: The type of plane, 1 for xy,2 for xz; integer, the variable number, i.e., 1 for u, 2 for v, 3 for w; integer, the coordinate of the plane; real, flow type `fltype`; integer, displacement thickness expressed in half box heights; real.

Record 4: Time; real, the length by which the boxed has been shifted to the right since time zero; real.

Record 5: The velocity array in physical space; x-y-planes are `nx`×`nyp` with `x` varying the fastest; x-z-planes are `nx`×`nz` for the non-symmetric case and `nx`×`(nz/2+1)` for the symmetric case with `x` varying the fastest.

Record 4-5 are repeated for each time when the plane is saved.

5.9. *xy-statistics file*

Unformatted, sequential.

Record 1: Reynolds number; real, `.false.` (this is to be backward compatible with channel flow files); logical, x_L; real, z_L; real, the time for this field; real, the length by which the boxed has been shifted to the right since time zero; real.

Record 2: Number of spectral modes in the x-direction; integer, the number of points in the physical y-direction; integer, the number of spectral modes in the z-direction reduced for symmetry; integer, 0/1 no z-symmetry/z-symmetry; integer.

Record 3: Flow type `fltype`; integer, displacement thickness expressed in half box heights; real.

Record 4: If `fltype` ≥ 4 : start of blending region `bstart`; real , end of blending region `bslope`; real , if `fltype` ≥ 7 acceleration exponent of streamwise free-stream velocity `rlam`; real, spanwise free-stream velocity `spanv`; real. For other values of `fltype` this record is omitted.

Record 5. Sum of the length of the time steps at which statistics have been sampled `sumw`; real, number of statistics calculated `nxys`; integer.

Record 6-5+`nxys`. Each record contains a `nx` $\times$ `nyp` plane of statistics with the x-index varying the fastest. The statistics are averaged over time and the z-direction.

Record 6-11 u, v, w, u^2, v^2, w^2.

Record 12-17 $\omega_1, \omega_2, \omega_3, \omega_1^2, \omega_2^2, \omega_3^2$

Record 18-20 uv, uw, vw

Record 21-23 $u(x)u(x+1), v(x)v(x+1), w(x)w(x+1)$ (i.e. one point separation auto correlations, x counted cyclically).

Record 24-26 $u(y)u(y+1), v(y)v(y+1), w(y)w(y+1)$

Record 27-29 $u(z)u(z+1), v(z)v(z+1), w(z)w(z+1)$ (z counted cyclically)

Record 30 $R\epsilon_{11} = u_x^2 + u_y^2 + u_z^2$, ϵ_{ij} is the dissipation tensor

Record 31 $R\epsilon_{22} = v_x^2 + v_y^2 + v_z^2$

Record 32 $R\epsilon_{33} = w_x^2 + w_y^2 + w_z^2$

Record 33 $R\epsilon_{12} = u_x v_x + u_y v_y + u_z v_z$

Record 34 $R\epsilon_{13} = u_x w_x + u_y w_y + u_z w_z$

Record 35 $R\epsilon_{23} = v_x w_x + v_y w_y + v_z w_z$

Record 36-47 $p, p^2, pu, pv, pw, pux, pvy, pwz, puy, pvx, upx, wpz$

5.10. *Free-stream velocity table file*

formatted, sequential

Record 1: `n` number of table entries

Record 2 - `n`+1: `xtab` streamwise coordinate; real, `utab` free-stream velocity; real.

5.11. **wave.d** *forced wave file*

formatted, sequential

Record 1: `rew` Reynolds number of wave (not used by **bla**); real.

Record 2: `alfaw` the streamwise, `betaw` the spanwise wavenumber of the wave; both real.

Record 3: `eig` the eigenvalue of the wave, the real part of which is used as the angular frequency of the wave; complex.

Record 4-`n`+3: `n` chebyshev coefficients of the mode shape of the normal velocity, of which the first `nyp` are used. If there are not enough coefficients they are padded by zeroes; complex.

5.12. **basic.i** *Base flow profile file*

basic.i is unformatted and sequential. **basic.i** is an output file from **cbflow**. **basic.i** saves the basic flow profile only for non-parallel spatial simulations if the file does not exist, or reads the basic flow profile for the same simulation parameters.

Record 1: Reynolds number; real, x_L; real, the length by which the boxed has been shifted to the right since time zero; real, displacement thickness expressed in half box heights `dstar`; real, start of blending region `bstart`, end of blending region `bslope`, acceleration exponent of streamwise free-stream velocity `rlam`, spanwise free-stream velocity `spanv`, the number of points in the physical x-direction; integer, the number of points in the physical y-direction; integer.

Record 2: The basic u, v, w-velocities in the physical x, y space.

Acknowledgments

The authors thank The Aeronautical Research Institute of Sweden (FFA) for generous financial support during the code development.

References

Alvelius, K. & Skote, M. 2000 The performance of a spectral simulation code for turbulence on parallel computers with distributed memory. *Tech. Rep.* TRITA-MEK 2000:17. Royal Institute of Technology, Stockholm.

Bech, K. H., Henningson, D. S. & Henkes, R. A. W. M. 1998 Linear and non-linear development of localized disturbances in zero and adverse pressure gradient boundary layers. *Phys. Fluids* **10**, 1405–1418.

Berlin, S., Hanifi, A. & Henningson, D. S. 1998*a* The neutral stability curve for non-parallel boundary layer flow. In Berlin, S. 1998, Oblique waves in boundary layer transition, Ph.D. Thesis, Department of Mechanics, KTH, Stockhom, TRITA-MEK 1998:7.

Berlin, S. & Henningson, D. S. 1999 A nonlinear mechanism for receptivity of free-stream disturbances. *Phys. Fluids* **11**, 3749–3760.

Berlin, S., Kim, J. & Henningson, D. S. 1998*b* Control of oblique transition by flow oscillations. *Tech. Rep.* TRITA-MEK 1998:6. Royal Institute of Technology, Stockholm.

Berlin, S., Lundbladh, A. & Henningson, D. S. 1994 Spatial simulations of oblique transition. *Phys. Fluids* **6**, 1949–1951.

Berlin, S., Wiegel, M. & Henningson, D. S. 1999 Numerical and experimental investigation of oblique boundary layer transition. *J. Fluid Mech.* **393**, 23–57.

Bertolotti, F. P., Herbert, T. & Spalart, P. R. 1992 Linear and nonlinear stability of the Blasius boundary layer. *J. Fluid Mech.* **242**, 441–474.

Canuto, C., Hussaini, M. Y., Quarteroni, A. & Zang, T. A. 1988 *Spectral Methods in Fluids Dynamics*. Springer.

Elofsson, P. A. & Lundbladh, A. 1994 Ribbon induced oblique transition in plane poiseulle flow. In *Bypass transition - proceedings from a mini-workshop* (ed. D. S. Henningson), pp. 29–41. TRITA-MEK Technical Report 1994:14, Royal Institute of Technology, Stockholm, Sweden.

Greengard, L. 1991 Spectral integration and two-point boundary value problems. *SIAM J. Numer. Anal.* **28**, 1071–1080.

Henningson, D. S. 1995 Bypass transition and linear growth mechanisms. In *Advances in turbulence V* (ed. R. Benzi), pp. 190–204. Kluwer Academic Publishers.

Henningson, D. S., Johansson, A. V. & Lundbladh, A. 1990 On the evolution of localized disturbances in laminar shear flows. In *Laminar-Turbulent Transition* (eds. D. Arnal & R. Michel), pp. 279–284. Springer-Verlag.

Henningson, D. S. & Lundbladh, A. 1994 Transition in Falkner-Skan-Cooke flow. *Bull. Am. Phys. Soc.* **39**, 1930.

Henningson, D. S. & Lundbladh, A. 1995 Evaluation of turbulence models from direct numerical simulations of turbulent boundary layers. FFA-TN 1995-09, Aeronautical Research Institute of Sweden, Bromma.

Henningson, D. S., Lundbladh, A. & Johansson, A. V. 1993 A mechanism for bypass transition from localized disturbances in wall bounded shear flows. *J. Fluid Mech.* **250**, 169–207.

Hildings, C. 1997 Simulations of laminar and transitional separation bubbles. *Tech. Rep.*. Department of Mechanics, The Royal Institute of Technology, Stockholm, Sweden.

Högberg, M. & Henningson, D. S. 1998 Secondary instability of cross-flow vortices in Falkner-Skan-Cooke boundary layers. *J. Fluid Mech.* **368**, 339–357.

Kim, J., Moin, P. & Moser, R. 1987 Turbulence statistics in fully developed channel flow. *J. Fluid Mech.* **177**, 133–166.

Kleiser, L. & Schumann, U. 1980 Treatment of incompressibility and boundary conditions in 3-D numerical spectral simulations of plane channel flows. In *Proc. 3rd GAMM Conf. Numerical Methods in Fluid Mechanics* (ed. E. H. Hirschel), pp. 165–173. Vieweg. Braunschweig.

Kreiss, G., Lundbladh, A. & Henningson, D. S. 1994 Bounds for threshold amplitudes in subcritical shear flows. *J. Fluid Mech.* **270**, 175–198.

Lu, Q. & Henningson, D. S. 1990 Subcritical transition in plane Poiseuille flow. *Bull. Am. Phys. Soc.* **35**, 2288.

Lundbladh, A. 1993 Growth of a localized disturbance in inviscidly stable shear flow. TRITA-MEK 93-04, Royal Institute of Technology, Stockholm, Sweden.

Lundbladh, A. & Henningson, D. S. 1993 Numerical simulation of spatial disturbance development in rotating channel flow. FFA-TN 1993-30, Aeronautical Research Institute of Sweden, Bromma.

Lundbladh, A., Henningson, D. S. & Johansson, A. V. 1992*a* An efficient spectral integration method for the solution of the Navier-Stokes equations. FFA-TN 1992-28, Aeronautical Research Institute of Sweden, Bromma.

Lundbladh, A., Henningson, D. S. & Reddy, S. C. 1994*a* Threshold amplitudes for transition in channel flows. In *Transition, Turbulence and Combustion, Volume I* (eds. M. Y. Hussaini, T. B. Gatski & T. L. Jackson), pp. 309–318. Dordrecht: Kluwer.

Lundbladh, A. & Johansson, A. V. 1991 Direct simulation of turbulent spots in plane Couette flow. *J. Fluid Mech.* **229**, 499–516.

Lundbladh, A., Johansson, A. V. & Henningson, D. S. 1992*b* Simulation of the breakdown of localized disturbances in boundary layers. Proceedings of the 4th European Turbulence Conference, Delft, The Netherlands.

Lundbladh, A., Schmid, P. J., Berlin, S. & Henningson, D. S. 1994*b* Simulation of bypass transition in spatially evolving flows. Proceedings of the AGARD Symposium on Application of Direct and Large Eddy Simulation to Transition and Turbulence, AGARD-CP-551.

Malik, M. R., Zang, T. A. & Hussaini, M. Y. 1985 A spectral collocation method for the Navier-Stokes equations. *J. Comp. Phys.* **61**, 64–88.

Nordström, J., Nordin, N. & Henningson, D. S. 1999 The fringe region technique and the fourier method used in the direct numerical simulation of spatially evolving viscous flows. *SIAM J. Sci. Comp.* **20** (4), 1365–1393.

Reddy, S. C., Schmid, P. J., Bagget, P. & Henningson, D. S. 1998 On stability of streamwise streaks and transition thresholds in plane channel flows. *J. Fluid Mech.* **365**, 269–303.

Schmid, P. J. & Henningson, D. S. 1992 A new mechanism for rapid transition involving a pair of oblique waves. *Phys. Fluids A* **4**, 1986–1989.

Schmid, P. J. & Henningson, D. S. 1993 Nonlinear energy density transfer during oblique transition in plane Poiseuille flow. *Tech. Rep.* TRITA-MEK 1993:5. Royal Institute of Technology, Stockholm.

Schmid, P. J., Lundbladh, A. & Henningson, D. S. 1994 Spatial evolution of disturbances in plane Poiseuille flow. In *Transition, Turbulence and Combustion, Volume I* (eds. M. Y. Hussaini, T. B. Gatski & T. L. Jackson), pp. 287–297. Dordrecht: Kluwer.

Schmid, P. J., Reddy, S. C. & Henningson, D. S. 1996 Transition thresholds in boundary layer and channel flow. In *Advances in Turbulence VI* (eds. S. Gavrilakis, L. Machiels & P. A. Monkewitz), pp. 381–384. Kluwer Academic Publishers.

Skote, M., Henkes, R. A. W. M. & Henningson, D. S. 1998 Direct numerical simulation of self-similar turbulent boundary layers in adverse pressure gradients. *Flow, Turbulence and Combustion* **60**, 47–85.

Spalart, P. R. & Yang, K. 1987 Numerical study of ribbon induced transition in blasius flow. *J. Fluid Mech.* **178**, 345–365.

Appendix A. Release notes

This manual refers to the following programs and packages :

bla v3.3

bls v1.8

rit v1.9

pre v1.0

ritpre v1.0

rps v1.13

cmp v1.9

fou v1.4

pxyst v1.5

pamp1 v1.1

pamp2 v1.3

pampw v1.1

pext1 v1.1

dfc v1.1

dpc v1.1

plot1 v1.7

VECFFT v1.1

dclib v1.4

fsdf v1.2

This is software which is distributed free on a limited basis; it comes with no guarantees whatsoever. Problems can be reported to henning@mech.kth.se or hnd@ffa.se, but no action is promised. If results obtained by using these programs are published the authors would like an acknowledgment.

Distribution of the code is done by email using a `uuencode`ed, `compress`ed `tar` file. A complete directory structure including all of the material above can be obtained by executing the following commands on the saved mail file, preferably called `prog`

```
uudecode   prog
uncompress prog.tar.Z
tar -xf    prog.tar
```

Makefiles appropriate for compiling the codes are also included for those using the UNIX operating system.

A version of **bla** (**blap** v1.0) exists that runs on computers with distributed memory. This version is slightly different from the one described in this report, and its efficiency has been thoroughly tested by Alvelius & Skote (2000).

Appendix B. Scaling of variables

We have chosen a scaling for all parameters based on the displacement boundary layer thickness and free-stream velocity at $t = 0$, $x = 0$ for the reference or base flow. However, internally in the simulation code **bla** the implementation uses a scaling based on the half box height. (The external and internal velocity scale is the same.) This means that all external data must be rescaled when read into the program, and the reverse scaling applied on output. If we let `dstar` be the displacement thickness expressed in half box heights, then the following scaling relationships hold:

time(internal)=time(external)*dstar

length(internal)=length(external)*dstar

velocity(internal)=velocity(external)

vorticity(internal)=vorticity(external)/dstar

force(internal)=force(external)/dstar

All formatted input and output files except the wave amplitude file use external scaling, whereas the unformatted files and the wave amplitude file use internal scaling.

Appendix C. Investigation of the fringe method

In some flow cases with large growth rates, e.g. flows with adverse pressure gradients and separation bubbles, a badly chosen fringe might not offer sufficient damping. The present study aims to give guidelines to choose an optimum fringe. Three types of flow have been studied, channel flow, boundary-layer flow with zero pressure gradient and boundary-layer flow with an adverse pressure gradient.

Although this report does not contain a description of the channel flow code, we include this flow case in the fringe investigation. This is done since it makes it possible to exemplify the properties of the fringe only related to the damping of disturbances, excluding the large forcing needed to return the mean or basic flow to its required inflow state.

The main parameters deciding the damping properties of the fringe are

- Length of Fringe (L)
- Strength of Fringe (λ)
- Shape of Fringe
- Resolution
- Influence of Blending (For Boundary-Layer)

Variations in all of these parameters have been made, with the main focus on the length and strength of the fringe. The shape of the fringe, i.e. how λ is varied in the fringe region, is of some importance. To simplify the investigation and reduce the number of variables it was decided to use a fringe where the strength is gradually increased until a maximum is reached and then immediately decreased to zero. This way only two variables describe the shape of the fringe, see figure 2. Generally the rise has been three fourths of the total length and the decrease of fringe strength one fourth of the length. The maximum strength is what will be denoted with λ hereafter. The gradual change of strength of the fringe is done with a smooth step function that has continuous derivatives of all orders, equation (114). Throughout this investigation the damping has been measured as the difference in amplitude of the disturbance when going into the fringe compared with the value going out of the fringe. All calculations were continued until the disturbance had been convected through the computational domain more than once, thus ensuring that a steady state was reached.

C.1. *Channel flow*

In the channel flow calculations, a fixed physical box of length 80 $h/2$ was used and thus the length of the computational box was varied when the length of

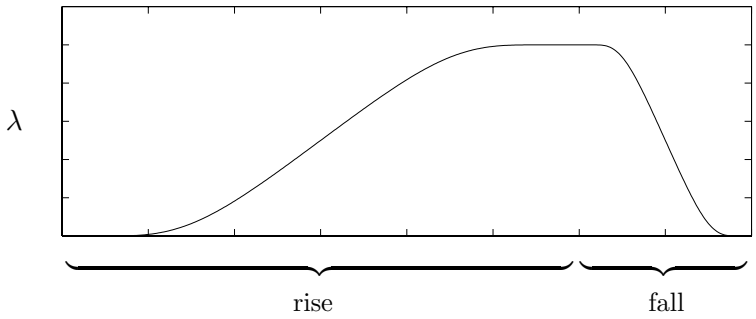

FIGURE 2. Schematic picture of the fringe used in the investigations. For this fringe the sum of the rise and fall is the same as the total length.

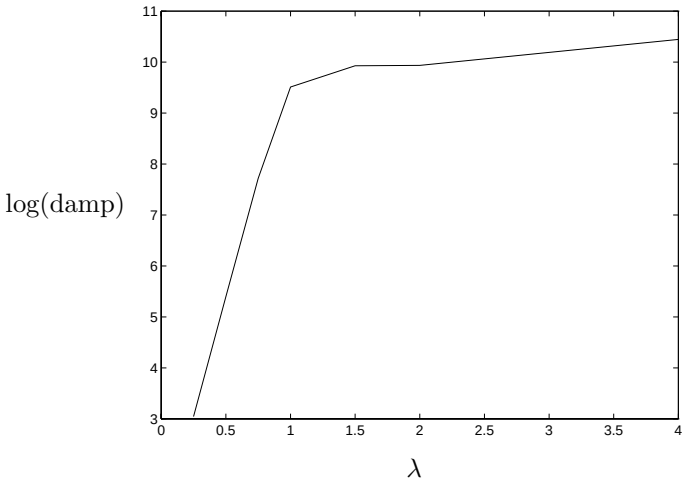

FIGURE 3. Damping as function of λ for channel flow.

the fringe changed. The Reynolds number based on the channel half height and centerline velocity was 3000 for all computations. To obtain the results shown in figures 3 and 4 a periodic volume force located at $x = 30$ with $\omega = 0.3$ has been used to introduce a disturbance that then evolved downstream. Figure 3 shows the damping as function of λ for a fixed length of the fringe. Note that the damping increases very rapidly with λ until it reaches a certain level from where further increase in damping is very modest. It is obvious that the strength integrated over the length of the fringe plays a major role of the damping. In figure 4 this is shown in a different way. Contours of the damping are plotted as function of the length and strength of the fringe region. For a given integral of the fringe region it is however advantageous to have a longer fringe with a lower λ.

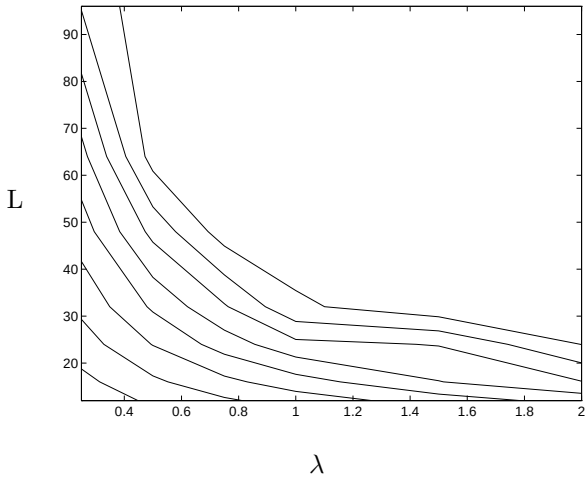

FIGURE 4. Contours of damping as function of the length of the fringe and λ. Each contour represents a magnitude of amplitude, from 2 to 9. Picture for channel flow.

In figure 5 the damping is shown as function of $\alpha = 2\pi/\sigma$, where σ is the wavelength, for three different frequencies. In all the cases the same fringe parameters have been used. The curves are obviously very close. This implies that without sufficient resolution the fringe cannot damp disturbances efficiently regardless of how well the fringe parameters are chosen. There is however an upper limit of the fringe damping regardless of the resolution. It is desirable to be close or at least know where this limit is. Based on this investigation one should strive for αdx to be approximately 0.5, i.e. $\sigma/dx = 2\pi/0.5 \approx 12.5$. This is a very high value for optimum performance of the fringe, it is however not likely that the highest frequencies are particularly amplified in other parts of the computational box and thus need the best damping. It is also possible that other parts of the flow require better resolution than the fringe, in which case the above requirement would not determine the necessary resolution.

C.2. *Boundary-Layer Flow*

In boundary-layer geometry the forcing is gradually varied from the corresponding outflow boundary-layer to the desired inflow. This variation in the forcing function is accomplished by the blending. The blending is achieved by varying the streamwise component of the velocity toward which the solution is forced according to

$$u_1^*(x,y) = U(x,y) + \Big[U(x + x_{period}, y) - U(x,y)\Big] S\left(\frac{x - x_{start}}{x_{rise}}\right), \qquad (113)$$

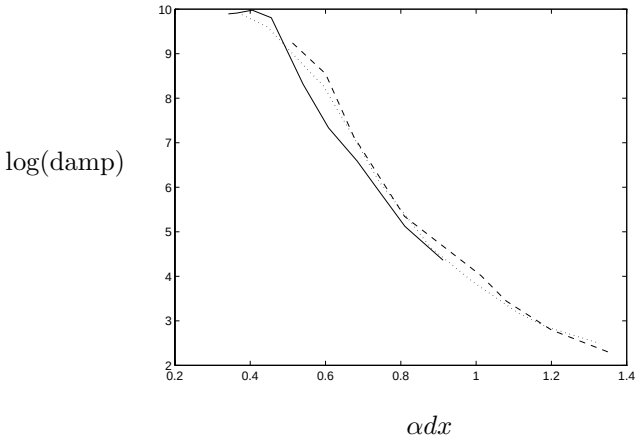

FIGURE 5. Damping as function of resolution for three different angular frequencies, $\omega = 1$ solid line, $\omega = 1.65$ dotted line and $\omega = 3$ dashed line, where $\alpha = 2\pi/\sigma$. Result for channel flow.

where $U(x, y)$ typically is a solution to the boundary-layer equations, x_{period} the streamwise length of the simulation box and

$$S(x) = \begin{cases} 0 & x \leq 0 \\ 1/[1 + \exp(\frac{1}{x-1} + \frac{1}{x})] & 0 < x < 1 \\ 1 & x \geq 1 \end{cases} . \tag{114}$$

The wall normal component of the velocity toward which the solution is forced is calculated from continuity.

Tests showed that the blending is of little importance for the damping. The blending should therefore be determined for maximum computational efficiency. If the flow is laminar or almost laminar the longest possible blending should be used, as the greatest gradients of the flow are likely to appear due to the blending, and thus regulate the finest resolution. If on the other hand the flow is turbulent, the largest gradients are usually in other parts of the domain and the resolution requirements due to the flow in the fringe are of less importance, allowing both shorter fringe and blending. It is worth noting that the profiles which the flow is forced towards are generally not solutions of the Navier-Stokes equations, rather these are usually similarity solutions of the boundary-layer equations.

For the calculations of the zero pressure-gradient boundary-layer flow a physical box of length 400 δ_0^* ($\delta_0^* = \delta^*$ at inflow) and height 10 δ_0^* was used. The Reynolds number was 1000 and the disturbance was introduced at $x = 200$ δ_0^* with $\omega = 0.1$

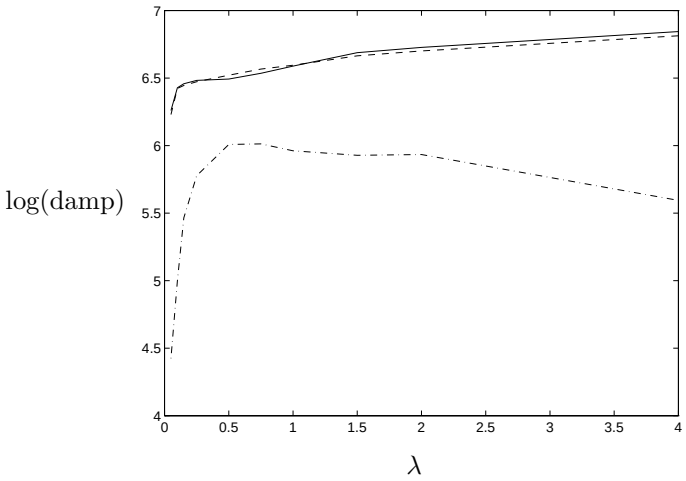

FIGURE 6. Damping as function of λ for three different boundary conditions. Solid line for bc prescribing the normal derivatives, dashed line for bc prescribing the streamwise velocity and dash-dotted line for the asymptotic condition. Length of fringe 400, with 75 % used for rise and 25 % for fall of the fringe function. Note that the damping is less than for channel flow.

For boundary-layer flow, there are differences in the fringe damping depending on the boundary condition used. Three different boundary conditions have been used in this investigation, the condition prescribing the normal derivatives, 101 in table 2, the asymptotic condition, 110 in table 2, and finally the boundary condition prescribing the streamwise velocity, 150 in table 2. In figure 6 the damping is plotted as function of λ for the different boundary conditions. The damping using the asymptotic boundary condition is somewhat less than the other two, and all three are smaller than in the channel flow calculations.

In figure 7 contours of damping are shown as function of strength and length of the fringe. The basic characteristics are the same as in the channel case. Figure 8 shows the effect of the shape of the fringe. The differences between the different cases are not large. This implies that there is only a small dependence on where the maximum strength of the fringe is reached, although the case with a very early maximum strength is the worst and should be avoided.

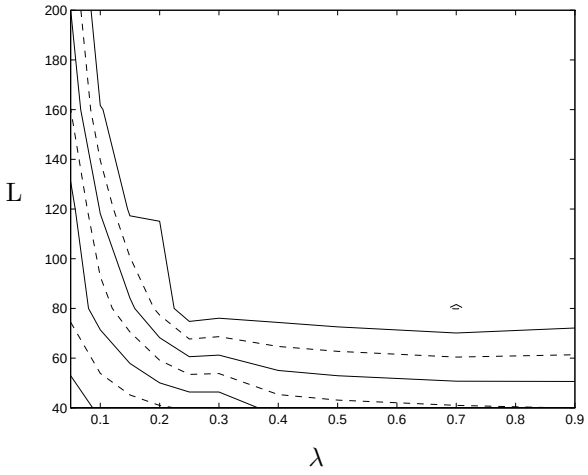

FIGURE 7. Damping as function of length of fringe and λ. Solid lines denote magnitudes of damping, from 2 to 5. The boundary condition prescribing the streamwise velocity has been used.

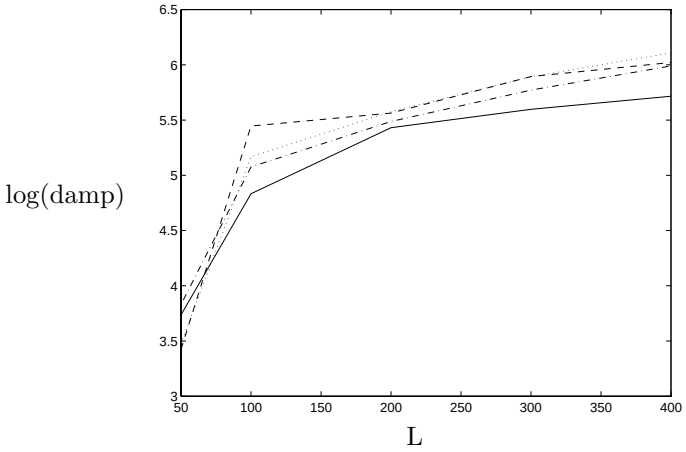

FIGURE 8. Damping as function of shape of fringe. Solid line corresponds to 25 % rise and 75 % fall. Dashed 50/50, dotted 75/25 and dash-dotted 100 % rise and zero distance for fall. The integral was held constant for the different lengths. The boundary condition prescribing the streamwise free-stream velocity, 150 in table 2, has been used.

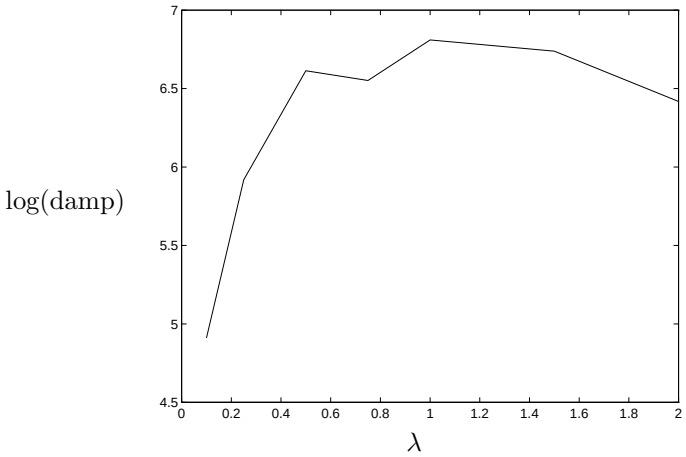

FIGURE 9. Damping as function of λ for boundary-layer flow with an adverse pressure gradient. Length of fringe was 200 δ_0^*. The boundary condition prescribing the streamwise velocity has been used.

C.3. *Boundary-Layer with Pressure Gradient*

C.3.1. *Qualities of the fringe*

For these calculations $Re_{\delta_0^*} = 1000$ and Hartree-Parameter $\beta = -0.18$ were chosen. Length of the physical part of the box was set to 200 δ_0^* and the height to 12. The blending started at the end of the physical box and used a rise distance of 100. A volume force with $\omega = 0.13$ was applied at $x = 100\ \delta_0^*$.

The main characteristics from the investigation with boundary layer flow are unchanged. However, there is now a much stronger natural amplification of disturbances. The same behavior of the damping as a function of λ as was observed for the investigation without pressure gradient is observed in figure 9. The total damping in the fringe is somewhat better than in the case without pressure gradient. In figure 10 contours of the damping are shown as function of the strength and length of the fringe. Quite surprisingly the damping deteriorates in some cases when the strength increases. The best damping is obtained with rather low values of λ.

C.3.2. *Spatial evolution of a disturbance*

The purpose of the remaining figures is mainly to show the evolution of a disturbance when it is convected through the computational box. Two different cases are studied, one without any forced disturbance and one with an introduced Tollmien-Schlichting wave. In figure 11 the frequency spectra at several

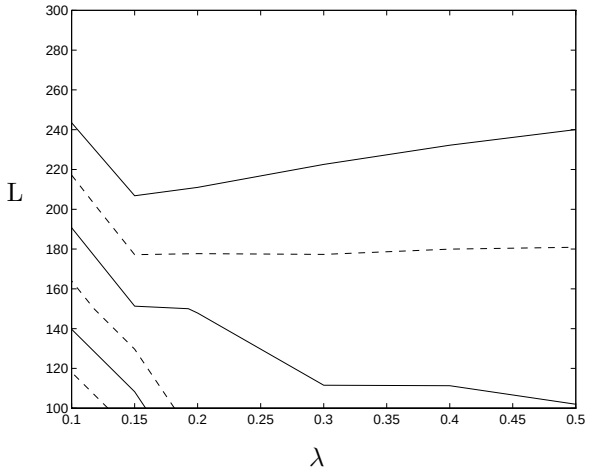

FIGURE 10. Contours of damping. Solid lines denote magnitudes of amplitude. The boundary condition prescribing the streamwise velocity has been used.

downstream positions are shown for the undisturbed case. Each curve shows the general disturbance level at the corresponding streamwise position. It is apparent that frequencies with $\omega = 0.1$ to 0.2 are the most amplified. The smallest disturbances are found at $x = 350$ and are mainly due to truncation errors. As the strength of the fringe decreases they start to grow. They reach their maximum intensity at $x = 200$, where they enter the fringe and are quickly damped. In figure 12 this evolution as well as that for the forced disturbance are shown, but only for the frequency that the TS-wave is forced with. Note that after the forcing the growth of the forced disturbance is greater than that of the unforced. It is also possible to see the upstream influence of the forcing. Of great importance is that the curve of the forced disturbance is above the curve of the undisturbed one. Figure 13 shows the evolution of the forced disturbance in the same manner as figure 12, i.e. the evolution for different frequencies at several streamwise positions. The greatest difference with the unforced case is the well defined peak at the frequency of the forcing. It is also possible to see that other frequencies than the forced one are the ones which grow at the end of the fringe.

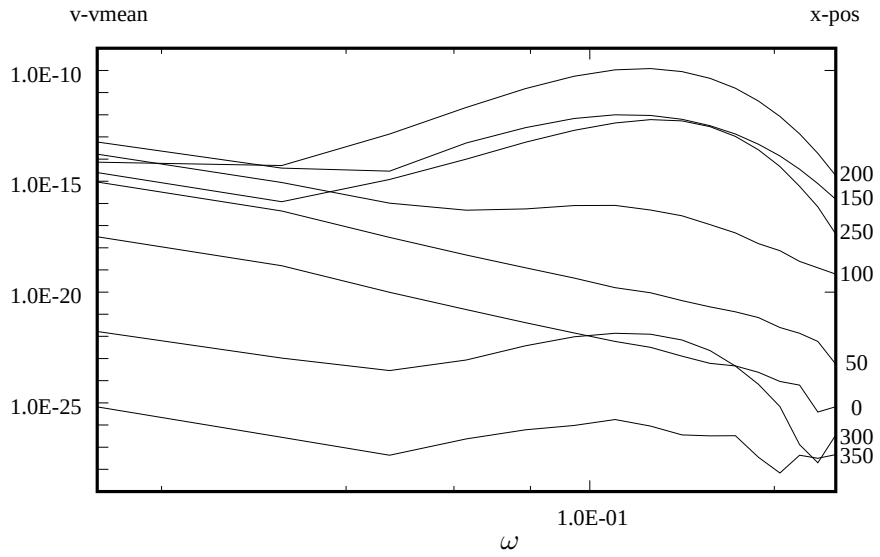

FIGURE 11. Frequency spectrum for different x-positions. For each line the x-position is given to the right. The fringe starts at $x = 200$ and total the length of the computational box is 400. No forcing is applied to create a disturbance, instead truncation errors grow in the physical part of the box.

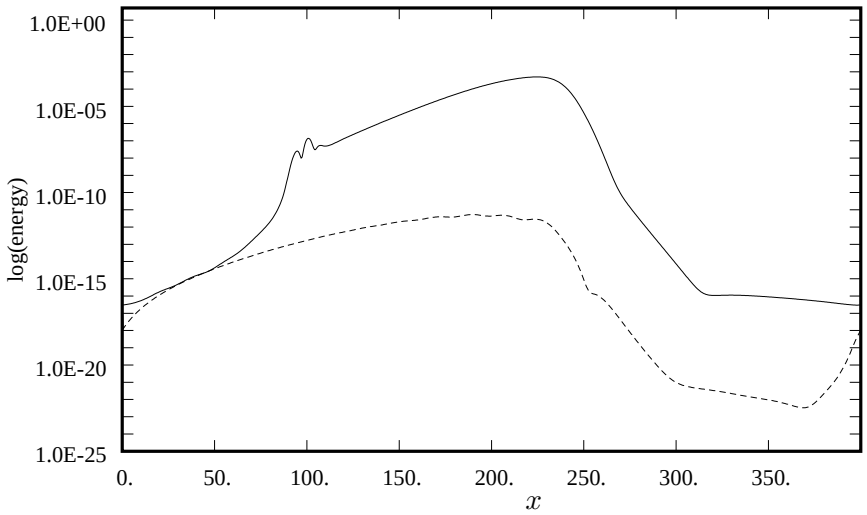

FIGURE 12. Energy of the forced wave as function of x. The solid line denotes a case where a volume force was introduced at $x = 100$. The dashed line represents the case of no forcing.

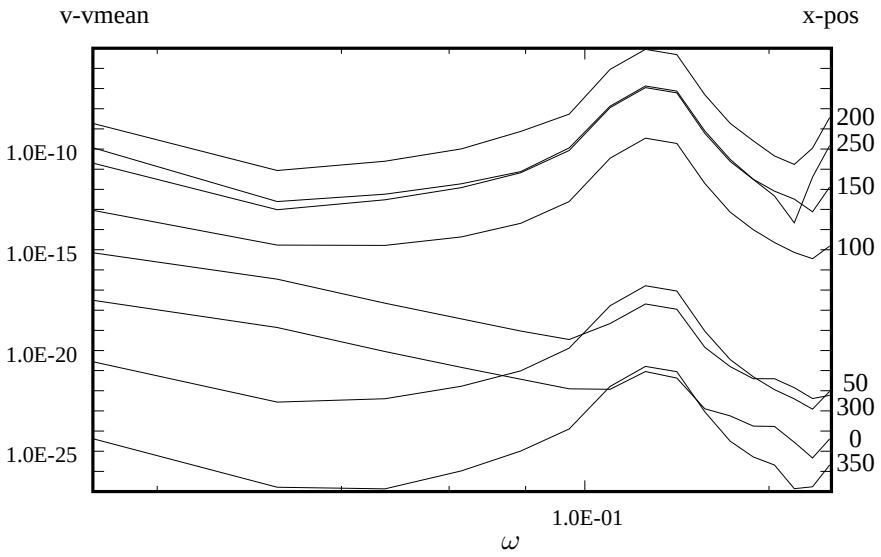

FIGURE 13. Frequency spectrum for different x-positions. For each line the x-position is given to the right. The fringe starts at $x = 200$ and total length of the computational box is 400. A volume force is applied at $x = 100$ with the frequency $\omega = 0.13$.

Appendix D. Examples, user created files

D.1. *Example* **par.f**, **bls.i**, **bla.i** *file for a simple simulation*

Below is an example of the adjustable part of a **par.f** include file. It is set up for a $32 \times 33 \times 32$ spectral mode simulation without spanwise symmetry, dealiazing in the x and z-direction, and in-core storage. The parameters `mby` and `mbz` are set for minimum storage.

```
c par.f contains size of problem
.
.
.
c adjustable parameters
c number of spectral modes
      parameter (nx=32,ny=33,nz=32)
c     dealiazing flags
      parameter (nfxd=1,nfyd=0,nfzd=1)
c symmetry flag
      parameter (nfzsym=0)
c core storage flag
      parameter (nfc=1)
c boxsize
      parameter(mby=1,mbz=1)
c number of processors
      parameter(nproc=1)
c bla with pressure solver (1)
c bls,rit,pre,cmp and bla without pressure (0)
      parameter(pressure=0)
c statistics
      parameter (nxys=42)
c computed parameters
.
.
.
```

Below is an example of a simple **bls.i** file to generate a localized disturbance in a file named **bl0.u**. Note that comments are allowed on lines with non-character data.

```
bl0.u
950.    re
100.    xl
10.     yl
50.     zl
3       fltype
0.      no Galilei shift velocity
.true.  generation of localized disturbance
1       type of disturbance
0.0002  amplitude
```

```
0.      rotation angle
2.      scale in x-direction
0.      origin in x-direction
2.      scale in y-direction
2.      scale in z-direction
1       type of distribution in the wall normal direction
.false. no waves
.false. no noise
```

Below is an example of a simple **bla.i** file to run initial data in file **bl0.u** to time 10 and output the result to file **bl10.u**. An amplitude list is written to bl10amp.d

```
bl0.u
bl10.u
10.             time for simulation
100             max iterations
7200.           max CPU time to stop(give a big value if not needed)
0.0             time step, =0 for automatic variation
4               number of time integration stages (1/3/4)
100.            keep old box length
.false.         no variable size
.0              rotation rate; no rotation
110             boundary condition at the free-stream
.false.         no chebyshev integration method; tau method
.false.         no Galilei transformation
.false.         no spatial simulation; temporal simulation
0.5             the boundary layer development speed
0               no localized volume force
.false.         no trip force
0               the boundary condition at the wall; no blowing/suction
4               cfl calc interval
4               amp calc interval
bl10amp.d
.false.         no y-dependent statistics
0               extremum calc interval; no extremum calc
0               xy-statistics calculation interval; no xy-stat calc
0               number of saved 3-d fields
0               number of saved wavenumbers
0               number of save planes
```

D.2. *Example* **par.f**, **bla.i** *file for a simulation of a turbulent boundary layer under an adverse pressure gradient.*

When running this example the turbulent statistics are stored in the file **endxys.u**. The simulation has to be run for a long time for the statistics to be sufficiently smooth. On a super computer the job can be restarted again after accomplishing a run. The different files for the statistics are then added together by the **addxys** program. The statistics are evaluated with the program **pxyst**. The velocity field **bl3400.u** and free-stream table **freestream.d015** are required when running this example.

```
c par.f contains size of problem
.
.
.
c adjustable parameters
c number of spectral modes
      parameter (nx=480,ny=161,nz=96)
c     dealiazing flags
      parameter (nfxd=1,nfyd=0,nfzd=1)
c symmetry flag
      parameter (nfzsym=0)
c core storage flag
      parameter (nfc=1)
c boxsize
      parameter(mby=2,mbz=2)
c number of processors
      parameter(nproc=6)
c bla with pressure solver (1)
c bls,rit,pre,cmp and bla without pressure (0)
      parameter(pressure=1)
c statistics
      parameter (nxys=42)
c computed parameters
.
.
.
```

Below is the **bla.i** file.

```
bl3400.u
bl3416.u
p3416.u
3416.            total simulation time
4000000          number of iterations
3600000.         cpu time
0.0              time step
4                1/3/4 number of stages
450.             keep old box length
.false.          variable size
.0               rotation rate
101              boundary condition number
.false.          no cim; use tau method
.false.          no Galilean transform
.true.           spatial simulation
.true.           read tabulated free-stream
freestream.d015
.false.          read in base flow; no base flow
1.25             strength of fringe region
-50.             start of fringe region
```

```
.0              end of fringe region
40.             rise distance of fringe
10.             fall distance of fringe
0.0             no oblique waves forced in the fringe
0.0             no two dimensional T-S wave
0               no localized volume force
.true.          trip forcing
0.0             steady forcing amplitude
0.2             time dependent forcing amplitude
4.0             x-length scale of trip
10.             x-origin of trip
1.0             y-length scale of trip
10              number of z-modes in trip
4.0             time-scale of trip
-1              random number seed for trip
0               the boundary condition at the wall; no blowing/suction
4               cfl calc interval
0               amp calc interval; no amplitude calculation
.false.         no y-dependent statistics
0               extremum calc interval; no extremum calculation
20              xy statistics calculation interval
endxys.u
50000           iterations between saves; do not save until finished
0.              time to start accumulation of statistics
3               number of saved 3-d fields
3404.
bl3404.u
3408.
bl3408.u
3412.
bl3412.u
0               number of saved wavenumbers
0               number of save planes
```