Xcompact3D Convergence Test

Paul Bartholomew

January 25, 2019

Abstract

This document details the setup, running and post-processing of the convergence test cases for `Xcompact3D`. These tests verify the spatial and temporal discretisations used by `Xcompact3D` using the 2D Taylor-Green Vortex (TGV) case as a benchmark. They are intended to run quickly on a desktop computer.

1 Introduction

`Xcompact3D` is developed to solve the Navier-Stokes equations, given for incompressible flow as

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla} \boldsymbol{u} = -\frac{1}{\rho} \boldsymbol{\nabla} p + \nu \boldsymbol{\Delta} \boldsymbol{u} \ , \tag{1}$$

$$\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0 \ , \tag{2}$$

where symbols have their usual meanings.

The 2D Taylor-Green Vortex, given by the initial conditions

$$u\left(x, y, 0\right) = \sin\left(x\right) \cos\left(y\right) \ , \tag{3}$$

$$v\left(x, y, 0\right) = -\cos\left(x\right) \sin\left(y\right) \tag{4}$$

has the solution

$$u\left(x, y, t\right) = e^{-2\nu t} \sin\left(x\right) \cos\left(y\right) \ , \tag{5}$$

$$v\left(x, y, t\right) = -e^{-2\nu t} \cos\left(x\right) \sin\left(y\right) \ , \tag{6}$$

and is simulated in the box $-\pi \leq x, y \leq \pi$.

By running at different spatial and temporal resolutions, the numerical results can be compared with the analytical one and the rate of convergence determined - to be checked with the theoretical rate of convergence for the chosen discretisation to confirm a correct implementation. This leads to a matrix of different test cases (including different boundary conditions for spatial convergence tests), identified by the scheme `sA-tB-bC-nD-dtE` for space, time, boundary conditions, number of mesh nodes and timestep. The boundary condition, mesh and timestep codes are self explanatory: `bC0012` corresponding to periodic in x, zero-flux at $y = -\pi$ and Dirichlet at $y = \pi$; `n128` indicates a 128×128 node mesh; and `dt0.001` indicates a timestep $\Delta t = 10^{-3}$. Similarly the space and time codes correspond to the values of `iorder` and `itimescheme` as indicated in table 1.

Table 1: Space and time code schema

Space code	Scheme	Time code	Scheme
s1	2^{nd} order central	t1	1^{st} order Euler
s2	4^{th} order central	t2	AB2
s3	4^{th} order compact	t3	AB3
s4	6^{th} order compact	t4	AB4 (not implemented)
		t5	RK3
		t6	RK4 (not implemented)

2 Running the tests

The same executable is used to run each test, to build this run (from the `Xcompact3D` project root directory):

```
make clean && \                                  # Ensure clean build
    make FLOW_TYPE=TGV TWOD=1 && \               # Build executable
    mv incompact3d examples/Test-Convergence/ # Move executable
```

The tests are run from the case directory: `${I3D_HOME}/examples/Test-Convergence/` using the command `make run` to run all test cases. To run a specific test suite you can specify the scheme (spatial or temporal) as `make run TEST=TYPE` where `TYPE` is one of the space or time codes given in table 1. Additionally, the number of processors to run on (default 1) can be set by passing `NP=X` to the `make` command.

2.1 input.i3d

A template `input.i3d` is given for the `s4t1b0000n128dt5e-4` case (the test scripts modify this as needed).

```
! -*- mode: f90 -*-

!====================
&BasicParam
!====================

! Domain decomposition
p_row = 0              ! Row partition
p_col = 0              ! Column partition

! Mesh
nx = 128               ! X-direction nodes
ny = 128               ! Y-direction nodes
nz = 8                 ! Z-direction nodes

istret = 0             ! y mesh refinement (0:no, 1:center, 2:both sides, 3:bottom)
beta = 0.3             ! Refinement parameter (beta)

! Domain
xlx = 6.28318530718    ! Lx (Size of the box in x-direction)
yly = 6.28318530718    ! Ly (Size of the boy in y-direction)
zlz = 6.28318530718    ! Lz (Size of the boz in z-direction)
```

```
! Flow parameters
itype = 8              ! Type of Flow
iin = 1                ! Inflow conditions (1: classic, 2: turbinit)
re = 1600.             ! nu=1/re (Kinematic Viscosity)
u1 = 8.                ! u1 (max velocity) (for inflow condition)
u2 = 8.                ! u2 (min velocity) (for inflow condition)
init_noise   = 0.0     ! Turbulence intensity (1=100%) !! Initial condition
inflow_noise = 0.0     ! Turbulence intensity (1=100%) !! Inflow condition

! Boundary conditions
nclx1 = 0
nclxn = 0
ncly1 = 0
nclyn = 0
nclz1 = 0
nclzn = 0

! Time stepping
dt = 0.0005            ! Time step
ifirst = 1             ! First iteration
ilast = 5000           ! Last iteration

! Enable modelling tools
iturbmod=0             ! if 0 then DNS
iscalar=0              ! If iscalar=0 (no scalar), if iscalar=1 (scalar)
iibm=0                 ! Flag for immersed boundary method

/End

!====================
&NumOptions
!====================
! Spatial derivatives
iorder = 4             ! (1->2nd central, 2->4th central, 3->4th compact, 4-> 6th
    mboxcompact)

! Time scheme
itimescheme = 1        ! Time integration scheme (1->Euler,2->AB2, 3->AB3, 4->AB4
    mbox,5->RK3,6->RK4)
/End

!=================
&InOutParam
!=================

! Basic I/O
irestart = 0           ! Read initial flow field ?
icheckpoint = 50000    ! Frequency for writing backup file
nvisu = 1              ! Size for visualisation collection
ioutput = 250          ! Frequency for visualization
/End

!=================
&Statistics
!=================
```

```
spinup_time = 50000.  ! Time after which statistics are collected (in seconds)
nstat = 1             ! Size arrays for statistic collection

/End
```

2.2 Spatial order of convergence

To test the spatial order of convergence, the mesh is systematically refined from 16^2 to 128^2 nodes. Each scheme should then converge to the correct solution with its theoretical order of accuracy - this can also be tested for different boundary conditions.

To measure the errors due to spatial discretisation, we must minimise the temporal error. The timestep used is $\Delta t = 5 \times 10^{-4}$, corresponding on the finest (128^2) mesh to $CFL \approx 0.01$, it should therefore suffice to use 1^{st} order Euler as the time discretisation (this has the simplest code to write and can be expected to be correct). The canonical TGV test case runs to non-dimensional time $t = 20$, with regards to running quickly on a typical desktop, here we run to $t = 2.5$, corresponding to $5,000$ timesteps[1].

The tests are run by selecting the script for the scheme you are interested in, *e.g.* `s4.sh` to test the 6^{th} order compact schemes, this cycles through each set of boundary conditions and resolution to test the implementation.

```
#
#         FILE: test-convergence.sh
#  DESCRIPTION: Runs convergence tests.
#       AUTHOR: Paul Bartholomew <paul.bartholomew08@imperial.ac.uk>
#

echo "============================================================="
echo " Running convergence tests for xcompact3d"
echo " Author: Paul Bartholomew <paul.bartholomew08@imperial.ac.uk>"
echo "============================================================="

CWD=$(pwd)

MESHES=( 16 32 64 128 )
SSCHEMES=( 1 2 3 4 )
FAILURES="The following tests failed to run:\n"

echo "Running case:"
for msh in "${MESHES[@]}"
do
    for sscheme in "${SSCHEMES[@]}"
    do
        if [ "${msh}" == "128" ]; then
            DELTAT=( 4e-3 2e-3 1e-3 5e-4 )
            TSCHEMES=( 1 2 3 4 )
        else
            DELTAT=( 5e-4 )
            TSCHEMES=( 1 )
        fi
        for tscheme in "${TSCHEMES[@]}"
```

[1] We know the solution as a function of t - we only need to run for enough time to exercise the code.

```
do
    if [ "${tscheme}" == "1" ]; then
        XBCS=( "00" "11" "22" "12" "21" )
        YBCS=( "00" "11" "22" "12" "21" )
    else
        XBCS=( "00" )
        YBCS=( "00" )
    fi
    for dt in "${DELTAT[@]}"
    do
        # Calculate number of steps to t=2.5
        NSTEP=$(echo "print(int(2.5 / ${dt}))" | python3)

        # Loop over boundary conditions
        for ncx in "${XBCS[@]}"
        do
            for ncy in "${YBCS[@]}"
            do
                # Set mesh size accoring to BCs
                if [ "${ncx}" = "00" ]; then
                    mshx=${msh}
                else
                    let mshx=${msh}+1
                fi
                if [ "${ncy}" = "00" ]; then
                    mshy=${msh}
                else
                    let mshy=${msh}+1
                fi

                # Setup working directory
                cd ${CWD}
                WORK=s${sscheme}/t${tscheme}/b${ncx}${ncy}/n${msh}/dt${dt}
                echo "- ${WORK}"
                mkdir -p ${WORK}
                cp input.i3d ${WORK}
                cp incompact3d ${WORK}
                cp probes.prm ${WORK}
                cp visu.prm ${WORK}
                cd ${WORK}

                # Modify input.i3d and run
                sed -i "s/nx = .*/nx = ${mshx} /g" input.i3d
                sed -i "s/ny = .*/ny = ${mshx} /g" input.i3d
                sed -i "s/dt = .*/dt = ${dt} /g" input.i3d
                sed -i "s/ilast = .*/ilast = ${NSTEP} /g" input.i3d
                sed -i "s/iorder = .*/iorder = ${sscheme} /g" input.i3d
                sed -i "s/itimescheme = .*/itimescheme = ${tscheme} /g"
                   mboxinput.i3d
                sed -i "s/nclx1 = .*/nclx1 = ${ncx:0:1} /g" input.i3d
                sed -i "s/nclxn = .*/nclxn = ${ncx:1:1} /g" input.i3d
                sed -i "s/ncly1 = .*/ncly1 = ${ncy:0:1} /g" input.i3d
                sed -i "s/nclyn = .*/nclyn = ${ncy:1:1} /g" input.i3d

                mpiexec -np 4 ./incompact3d > OUTPUT.log
                if [ "$?" != "0" ]; then
```

```
                            FAILURES="${FAILURES}${WORK}\n"
                        fi
                    done
                done
            done
        done
    done
done

echo "------------------------------------------------------------"
echo -e ${FAILURES}
```

2.3 Temporal order of convergence

To test the temporal order of convergence, the timestep is systematically refined in the range $5 \times 10^{-4} \leq \Delta t \leq 8 \times 10^{-3}$. The results can be similarly compared with the analytical solution and the order of convergence determined. As we are now interested in temporal error, these tests are run on the finest (128^2) mesh.

3 Analysing the results

To analyse the results we load for each level of refinement (spatial or temporal) the final result ($t = 2.5$) and compute the `RMS` error between the numerical and analytical result

$$\varepsilon = \sqrt{\frac{1}{N} \sum_{i}^{N} (\phi_i^{num.} - \phi_i^{an.})^2} \tag{7}$$

as computed by the following `python` code

```
def calc_rms(num, an):
  """Compute the RMS of the difference between a numerical and an analytical
     mboxresult.

  -- INPUT --
  - num: a 3D numpy array (loaded from incompact3d).
  - an: a 2D numpy array (generated analytically) on the 2D mesh."""

  nx = num.shape[0]
  ny = num.shape[1]
  n = nx * ny

  rms = 0

  rms = ((num[:,:,0] - an[:,:])**2).sum()
  rms /= n
  rms = rms**0.5

  return rms
```

To do the comparison we need to compute the analytical solution on our grid:

```
def sol_an(nx, ny, t=2.5, Re=1600.):
  """Compute the analytical solution of the 2D Taylor-Green vortex on a nx-by-ny
  grid at time t, Reynolds number Re."""

  u = np.zeros((nx, ny))
  v = np.zeros((nx, ny))
  dx = 2 * math.pi / (nx - 1)
  dy = 2 * math.pi / (ny - 1)

  F = math.exp(-2 * t / Re)
  for i in range(nx):
    x = i * dx
    for j in range(ny):
      y = j * dy

      u[i][j] = F * math.sin(x) * math.cos(y)
      v[i][j] = -F * math.cos(x) * math.sin(y)

  return u, v
```

To load the numerical data, we use `Py4Incompact3d`

```
def load(filepath, tstep=20, vars=["ux", "uy"]):

  # Generate a control dictionary
  gendict("input.json", filepath, vars)

  postproc = Postprcess("input.json")
  postproc.load(time=tstep)

  # Build return tuple (unpacks like regular variables)
  ret = ()
  for var in vars:
    ret = ret + (postprocess.fields[var].data[t],)

  return ret
```

3.1 Spatial order of convergence

3.2 Temporal order of convergence