Cgad User Guide: An Overture Solver for the Advection Diffusion Equations on Composite Grids

William D. Henshaw,
Department of Mathematical Sciences,
Rensselaer Polytechnic Institute ,
Troy, NY, USA, 12180.
`overtureFramework.org`

June 26, 2021

Abstract:
Cgad is a program that can solve the advection diffusion (and reaction) equations on overlapping grids. Cgad can solve problems on moving and deforming domains. Cgad can also be used with Cgmp to solve the heat equation in solid regions, when performing conjugate heat transfer problems in conjuction with the Cgins flow solver.

Contents

1 Introduction

Cgad is a program that can solve the advection diffusion equations on overlapping grids and is based on the Overture framework[1, 3, 2]. A block diagram of the main components of Overture is given in Figure 1. Overture provides support for solving PDEs on overset grids. More information about **Overture** can be found on the **Overture** home page, `www.overtureFramework.org`. Cgad can solve problems on moving and deforming domains. Cgad can also be used with Cgmp to solve the heat equation in solid regions, when performing conjugate heat transfer problems in conjuction with the Cgins flow solver (see Figure 2).

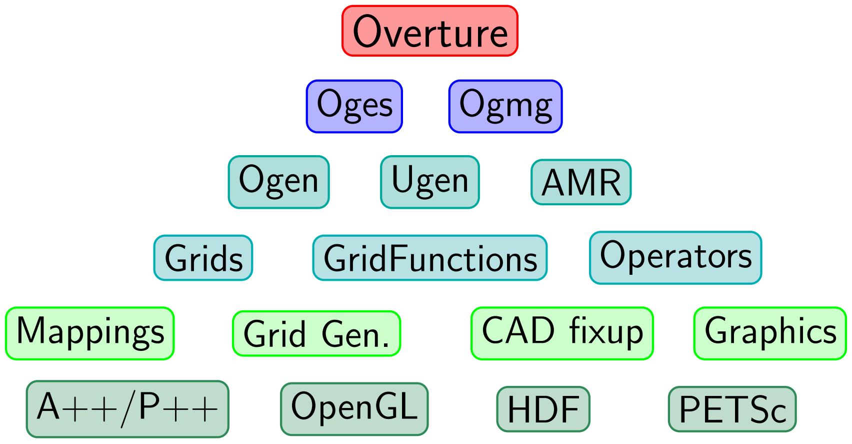

Figure 1: Overture block diagram. Overture supports the solution of PDEs on overset grids. It has classes to hold grids and grid functions. It has support for component grid generation and overset grid generation. Oges is the interface to sparse solvers such as those in PETSc. A++/P++ is the serial/parallel array class for C++.

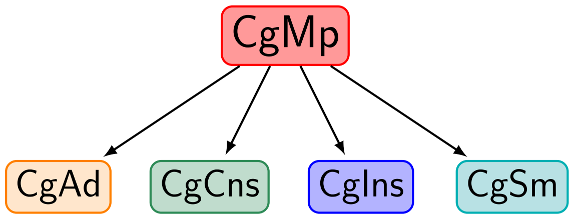

Figure 2: Cgmp drives different CG solvers to perform CHT and FSI simulations. CgAd solves for heat transfer in solids. CgCns solves compressible flow. Cgins solves incompressible flow. CgSm solves for deformations of solids, linear and nonlinear elasticity.

Cgad can solve for m unknowns dependent variables $u_j = u_j(\mathbf{x}, t)$, $j = 1, 2, \ldots, m$ where the *components* satisfy a potentially coupled system of *advection diffusion* (and reaction) equations of the form,

$$\frac{\partial u_j}{\partial t} + a_1(\mathbf{x}, t, \mathbf{u})\frac{\partial u_j}{\partial x} + a_2(\mathbf{x}, t, \mathbf{u})\frac{\partial u_j}{\partial y} + a_3(\mathbf{x}, t, \mathbf{u})\frac{\partial u_j}{\partial z} = \nabla \cdot \big(\kappa_j(\mathbf{x}, t, \mathbf{u})\nabla u_j\big) + f_j(\mathbf{x}, t, \mathbf{u}), \qquad (1)$$

where $a_k(\mathbf{x}, t, \mathbf{u})$, $k = 1, 2, 3$ are the advection coefficients, $\kappa_j = \kappa_j(\mathbf{x}, t, \mathbf{u})$ are the diffusion coefficients, and $f_j = f_j(\mathbf{x}, t, \mathbf{u})$ is a body forcing. Here $\mathbf{u} = [u_1, u_2, \ldots, u_m]^T$ denotes the vector with components u_j. Boundary conditions take the form of Dirichlet, Neumann or mixed (Robin).

Cgad features:

- supports multiple components,
- variable diffusivity coefficients $\kappa_j(\mathbf{x}, t, \mathbf{u})$ (that can depend on $\mathbf{x}$, t and the solution),
- variable advection coefficients, $a_k(\mathbf{x}, t, \mathbf{u})$,
- 2D axisymmetric option,

- user defined body forcing,
- user defined moving and deforming surfaces.

The variable diffusivity and advection coefficents can be specified through a *userDefined* function.

Installation: To install cgad you should follow the instructions for installing Overture and cg from the Overture web page. Note that cgad is NOT built by default when building the cg solvers so that after installing cg you must go into the `cg/ad` directory and type 'make'. If you only wish to build cgad and not any of the other cg solvers, then after building Overture and unpacking cg, go to the `cg/ad` and type make.
The cgad solver is found in the `ad` directory in the `cg` distribution and has sub-directories

`bin` : contains the executable, cgad. You may want to put this directory in your path.

`check` : contains regression tests.

`cmd` : sample command files for running cgad, see section (2).

`doc` : documentation.

`lib` : contains the cgad library, `libCgad.a`.

`src` : source files

`runs` : sample demonstrations of using Cgad to solve various problems.

1.1 Basic steps

Here are the basic steps to solve a problem with cgad.

1. Generate an overlapping grid with ogen.
2. Run cgad (found in the `bin/cgad` directory).
3. Assign the boundary conditions and initial conditions.
4. Choose the parameters for the PDE (such as material properties).
5. Choose run time parameters, time to integrate to, time stepping method etc.
6. Compute the solution (optionally plotting the results as the code runs).
7. When the code is finished you can look at the results (provided you saved a 'show file') using `plotStuff`.

The commands that you enter to run cgad can be saved in a command file (by default they are saved in the file 'cgad.cmd'). This command file can be used to re-run the same problem by typing 'cgad file.cmd'. The command file can be edited to change parameters.

To get started you can run one of the examples in the `cmd` directory or in the `runs` directory.

2 Sample command files for running cgad

Command files are supported throughout the Overture. They are files that contain lists of commands. These commands can initially be saved when the user is interactively choosing options. The command files can then be used to re-run the job. Command files can be edited and changed.

In this section we present a number of command files that can be used to run cgad.

2.1 Running a command file

Given a command file for cgad such as `heatSource.cmd`, found in `cmd/heatSource.cmd`, one can type '`cgad heatSource.cmd`' to run this command file . You can also just type '`cgad heatSource`, leaving off the `.cmd` suffix. Typing '`cgad -noplot heatSource`' will run without interactive graphics (unless the command file turns on graphics). Note that here I assume that the `bin` directory is in your path so that the `cgad` command is found when you type it's name. The Cgad sample command files will automatically look for an overlapping grid in the `Overture/sampleGrids` directory, unless the grid is first found in the location specified in the command file.

When you run a command file a graphics screen will appear and after some processing the run-time dialog should appear and the initial conditions will be plotted. The program will also print out some information about the problem being solved. At this point choose `continue` or `movie mode`. Section (**??**) describes the options available in the run time dialog.

When running in parallel it is convenient to define a shell variable for the parallel version of cgad. For example in the tcsh shell you might use

```
  set cgadp  = ${CGBUILDPREFIX}/ad/bin/cgad
```

An example of running the parallel version is then

```
  mpirun -np 2 $cgadp heatSource
```

2.2 A deforming domain example

The directory `cg/ad/runs/deform` holds an example *run* that demonstrates using Cgad to solve a problem on a deforming domain.

Sinusoid Deform Example

(1) Generate the grid using the command file `freeSurfaceGrid2d.cmd` in `Overture/sampleGrids`:

```
ogen -noplot freeSurfaceGrid2d -interp=e -factor=4 -ml=1
```

(2) Run cgad: Advection diffusion (AD) of a Gaussian pulse with a sinusoidally deforming surface.

```
cgad deformingSurface -g=freeSurfaceGrid2de4.order2.ml1 -motion=sinusoid -kappa=.02 ...
       -ampy=.05 -tf=2. -tp=.01 -ic=pulse -tz=none -go=halt
```

Results from this run are shown in Figure 3.

Notes:

1. Note that the command line arguments are processed and used in the .cmd file (they are not directly passed to cgadMain, but only indirectly passed).
2. The motion of the interface is defined in `cg/ad/userDefinedDeformingSurface.C`
3. To support deforming motion, the grid to be deformed should be constructed using a NurbsMapping to define the surface an the hyperbolic grid generator (as found in `freeSurfaceGrid2d`).

Concentration Deform Example This example demonstrates a simple case where the motion of the deforming surface depends on the solution. Use the same grid as for the *Sinusoid Deform Example.* (2) Run cgad: Advection diffusion (AD) of a Gaussian pulse with surface whose motion depends on the concentration.

```
cgad deformingSurface -g=freeSurfaceGrid2de4.order2.ml1 -motion=concentration -kappa=.05 ...
         -tf=10. -tp=.05 -tz=none -ic=pulse -a=0 -b=0 -bc=n -ts=im -go=halt
```

Results from this run are shown in Figure 4.

Notes:

1. the motion of the interface is defined in `cg/ad/userDefinedDeformingSurface.C` and satisfies the equation

$$\frac{\partial \eta(s,t)}{\partial t} = \alpha(u(s,t) - u_e)$$

   where $y = \eta(s,t)$ is the height of the interface, s is the arclength variable (for the undeformed interface), $u(\mathbf{x}(s),t)$ is the solution to the AD equations and where α and u_e are some specified constants. For $\alpha > 0$, the interface will move downward where the solution u on the interface is less than u_e, and move upward where u is greater than u_e.

2. Note that Neumann boundary conditions are used (-bc=n).
3. The option `-ts=im` causes an implicit time-stepping algortihm to be chosen.

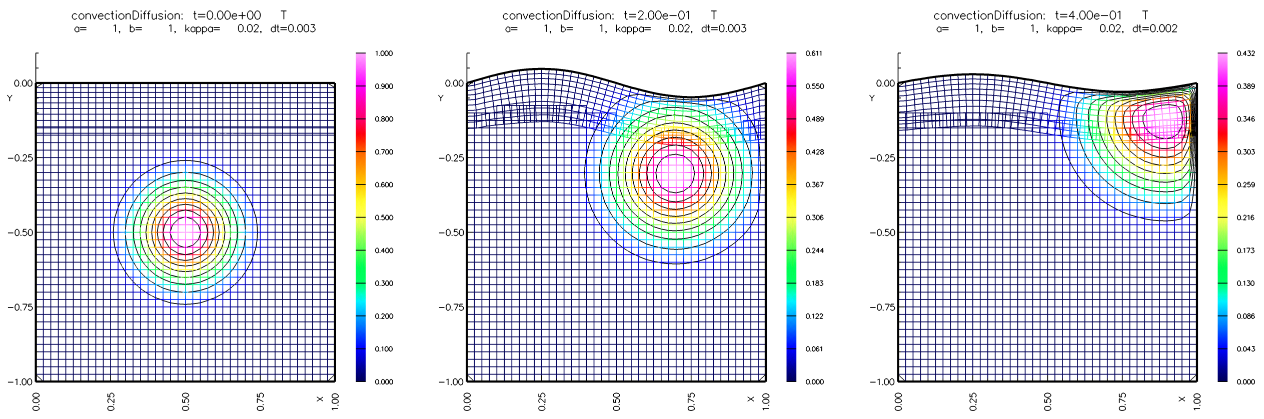
Figure 3: Results from the *Sinusoid Deform Example.* The top surface deforms over time with a sunusoidal motion as a pulse advects (upward and to the right) and diffuses.

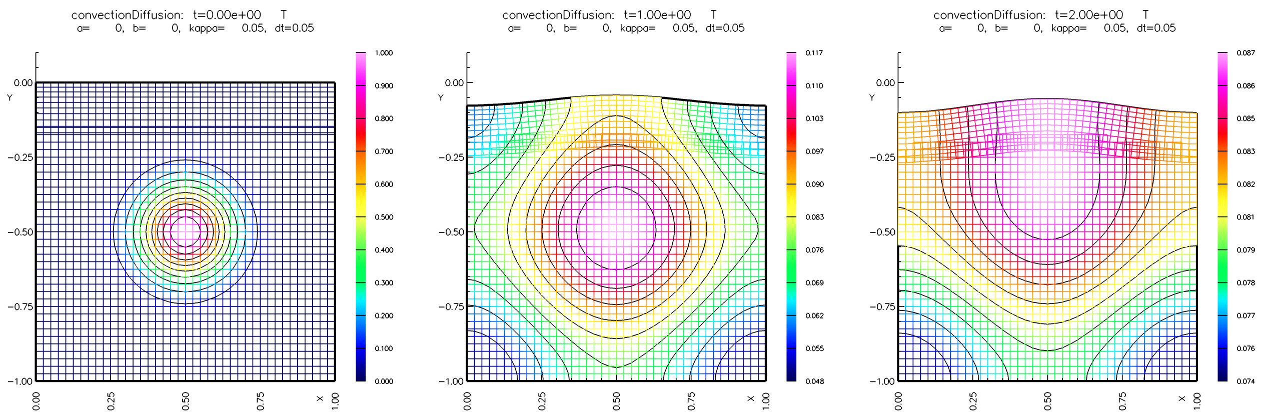
Figure 4: Results from the *Concentration Deform Example.* The top surface deforms over time as a function of the current concentration on the interface.

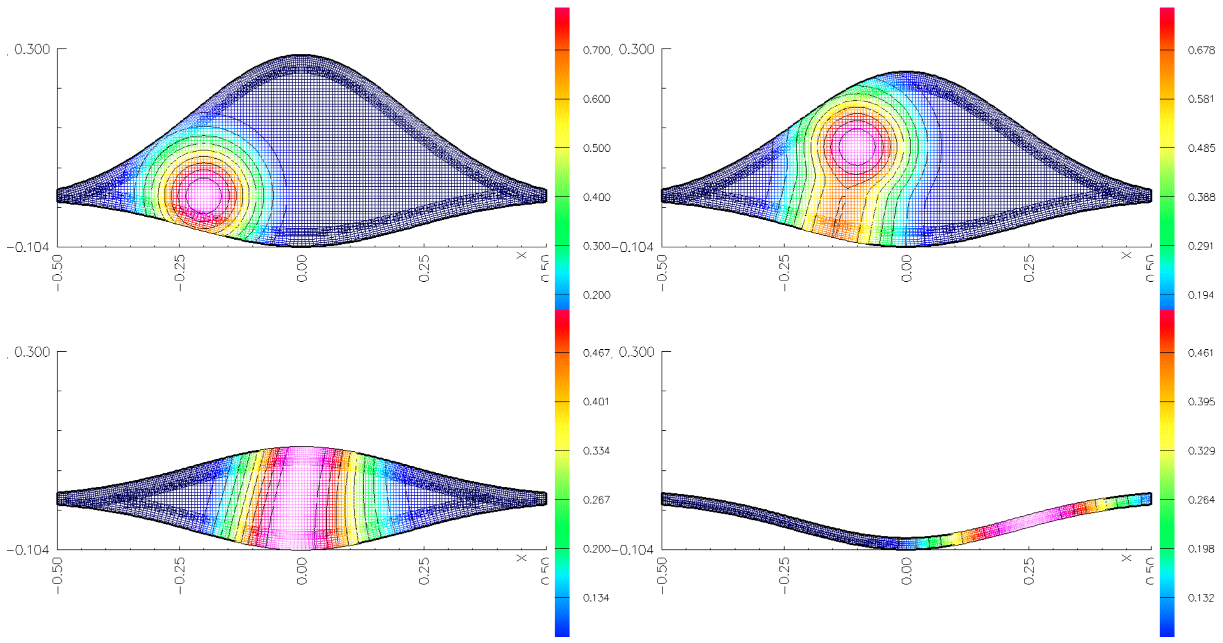

Figure 5: Results from the *deforming eye*. The top surface deforms over time by a specified motion.

2.3 User defined variable coefficients

A user can supply variable advection coefficients and variaable diffusion coefficients by editing the file `cg/ad/src/getCoefficients.C` and using the appropriate run time options.

2.4 User defined deforming surface

To define the motion of a deforming surface, one can add a new option to the file `cg/ad/src/userDefinedDeformingSurface.C`.

2.5 User defined forcing

The file `cg/common/src/userDefinedForcing.C` can be used to define new forcing functions for use with Cgad. Here are the steps to take to add your own forcing:

1. Edit the file `cg/common/src/userDefinedForcing.C`.
2. Add a new option to the function `setupUserDefinedForcing()`. Follow one of the previous examples.
3. Implement the forcing option in the function `userDefinedForcing(...)`.
4. Type *make* from the `cg/ad` directory to recompile Cgad.
5. Run Cgad and choose the *userDefinedForcing* option from the *forcing options...* menu. (see the example command file `ad/cmd/userDefinedForcing.cmd`).

3 Options

3.1 Options affecting the scheme

cfl *value* : set the CFL number. The schemes are usually stable to CFL=1. The default is CFL=.9.

3.2 Run time options

** FINISH ME **

tFinal *value* : solve the equations to this time.

tPlot *value* : time increment to save results and/or plot the solution.

debug *value* : an integer bit flag that turns on debugging information. For example, set debug=1 for some info, debug=3 (=1+2) for some more, debug=7 (=1=2+4)for even more.

3.3 Plotting options

** FINISH ME **

plot errors [0|1] :

check errors [0|1] :

3.4 Output options

** FINISH ME **

specify probes : specify a list of probe locations as x, y, z values, one per line, finishing the list with 'done'. The solution values at these locations (actually the closest grid point to each location, with these values written to the screen) are written to a text file whose default name is "probeFile.dat". In the following example we specify two probe locations, $(.2, .3, .1)$ and $(.4, .6, .3)$,

```
specify probes
  .2 .3 .1.
  .4 .6 .3
done
```

probe file: *name* : specify the name of the probe file.

probe frequency *value* : specify the frequency at which values are saved in the probe file. For example, if the probe frequency is set to 2 then the solution will be saved every 2nd time step to the probe file.

4 Examples and code verification

Here are some sample simulations. Some of these examples verify the accuracy of the approximations.

Notation:

> **FDm** - order m finite difference approximation in space.
> **PCm** - Adams predictor-corrector time-stepping, order m
> **BDFm** - BDF time-stepping, order m
> **CN** - Crank-Nicolson (trapezoidal rule) time-stepping, order 2

Note: For grid convergence studies with implicit time-stepping where there is no time-step restriction for stability, the time-step is chosen to scale like the grid spacing. In particular, the time-step on the coarsest grid is taken as the input parameter dtMax. Thus the time-step for grid resolution $m = 1, 2, 3, \ldots$, is taken as $\Delta t_m = \text{dtMax}/2^{m-1}$, assuming the grid spacing decreases by a factor or 2 at each new resolution.

4.1 Eigenfunctions of the heat equation on a square

The eigenfunctions for the heat equation on the square $[0,1]^2$ take the form

$$u(\mathbf{x},t) = a\, e^{-\beta t}\, \sin(k_x \pi x)\, \sin(k_y \pi y), \qquad \text{for Dirichlet BCs,} \tag{2}$$

$$u(\mathbf{x},t) = a\, e^{-\beta t}\, \cos(k_x \pi x)\, \cos(k_y \pi y), \qquad \text{for Neumann BCs,} \tag{3}$$

where k_x and k_y are integers and

$$\beta = \kappa(k_x^2 + k_y^2) \tag{4}$$

The form of the eigenfunctions in a 3D box are similar.

Here are some results from grid convergence studies. See `cg/ad/conv/Makefile` for commands to generate these tables.

PC2+FD2:

grid	N	u	r
square8	1	3.8e-3	
square16	2	9.7e-4	3.92
square32	4	2.4e-4	3.98
square64	8	6.1e-5	3.99
rate		1.99	

Table 1: Cgad, squareEigen, method=CGFD2, Max norm, method=NFDTD, known=squareEigen, ts=pc2, order=2, $t = .5$, cfl=0.9, kappa=.1, m=2, n=2 kx=2, ky=2, kz=1, tz=poly degreex=2 degreet=2 fx=0.5 fy=0.5 fz=0.5 -ft=0.5 Wed Apr 21 17:39:54 2021

PC2+FD4:

grid	N	u	r
square8	1	4.6e-4	
square16	2	3.1e-5	14.79
square32	4	9.8e-7	31.67
square64	8	2.2e-8	43.64
rate		4.79	

Table 2: Cgad, squareEigen, method=pc2, Max norm, method=pc2, known=squareEigen, ts=pc2, order=4, bdfOrder=2. $t = .5$, cfl=0.9, kappa=.1, m=2, n=2 kx=2, ky=2, kz=1, tz=poly degreex=2 degreet=2 fx=0.5 fy=0.5 fz=0.5 -ft=0.5 Thu Apr 22 07:40:49 2021

PC2+FD4:

grid	N	u	r
square8	1	4.6e-4	
square16	2	3.1e-5	14.79
square32	4	9.8e-7	31.67
square64	8	2.2e-8	43.64
rate		4.79	

Table 3: Cgad, squareEigen, method=pc2, Max norm, method=pc2, known=squareEigen, ts=pc2, order=4, bdfOrder=2. $t = .5$, cfl=0.9, kappa=.1, m=2, n=2 kx=2, ky=2, kz=1, tz=poly degreex=2 degreet=2 fx=0.5 fy=0.5 fz=0.5 -ft=0.5 Thu Apr 22 07:40:49 2021

BDF4+FD4:

grid	N	u	r
square8	1	8.3e-4	
square16	2	4.4e-5	18.60
square32	4	1.7e-6	25.96
square64	8	6.1e-8	28.26
square128	16	2.3e-9	26.77
rate		4.65	

Table 4: Cgad, squareEigen, method=bdf4, Max norm, method=bdf4, known=squareEigen, ts=bdf, order=4, bdfOrder=4. $t = .5$, cfl=0.9, kappa=.1, m=2, n=2 kx=2, ky=2, kz=1, tz=poly degreex=2 degreet=2 fx=0.5 fy=0.5 fz=0.5 -ft=0.5 Thu Apr 22 07:42:34 2021

4.2 Eigenfunctions of the heat equation on a disk

The eigenfunctions for the heat equation on the disk of radius $r = a$ take the form

$$u(\mathbf{x},t) = a\, e^{-\kappa\lambda^2 t}\, J_n(\lambda r)\, \sin(m\theta), \tag{5}$$
$$\tag{6}$$

where $\lambda = \lambda_{m,n}$ is the m'th root of

$$J_n(\lambda a) = 0, \qquad \text{for Dirichlet BC at } r = a, \tag{7}$$
$$J_n'(\lambda a) = 0, \qquad \text{for Neumann BC at } r = a, \tag{8}$$

Here are some results from grid convergence studies. See `cg/ad/conv/Makefile` for commands to generate these tables.

CN+FD2:

grid	N	u	r
sic2	1	9.9e-4	
sic4	2	2.5e-4	4.01
sic8	4	6.2e-5	4.01
sic16	8	1.5e-5	4.01
rate		2.00	

Table 5: Cgad, diskEigen, method=CGFD2, Max norm, method=NFDTD, known=diskEigen, ts=im, order=2, t = .5, cfl=0.9, kappa=.1, m=1, n=0 kx=1, ky=1, kz=1, tz=poly degreex=2 degreet=2 fx=0.5 fy=0.5 fz=0.5 -ft=0.5 Wed Apr 21 18:47:06 2021

BDF2+FD2:

grid	N	u	r
sic2	1	2.0e-3	
sic4	2	4.4e-4	4.55
sic8	4	1.0e-4	4.29
sic16	8	2.5e-5	4.10
rate		2.11	

Table 6: Cgad, diskEigen, method=bdf2, Max norm, method=bdf2, known=diskEigen, ts=bdf, order=2, bdfOrder=2, dtMax=.05, t = .5, cfl=0.9, kappa=.1, m=1, n=1 kx=1, ky=1, kz=1, tz=poly degreex=2 degreet=2 fx=0.5 fy=0.5 fz=0.5 -ft=0.5 Thu Apr 22 07:45:29 2021

BDF4+FD4:

grid	N	u	r
sic2	1	1.2e-4	
sic4	2	5.9e-6	19.69
sic8	4	3.3e-7	17.63
sic16	8	2.0e-8	16.78
rate		4.17	

Table 7: Cgad, diskEigen, method=bdf4, Max norm, method=bdf4, known=diskEigen, ts=bdf, order=4, bdfOrder=4, dtMax=.05, t = .5, cfl=0.9, kappa=.1, m=1, n=1 kx=1, ky=1, kz=1, tz=poly degreex=2 degreet=2 fx=0.5 fy=0.5 fz=0.5 -ft=0.5 Thu Apr 22 07:46:13 2021

4.3 Eigenfunctions of the heat equation on an annulus

In this section we describe some exact solutions for the heat equation, $u_t = \mathcal{D}\Delta u$, in an annular region. These solutions are implemented in `cgad/src/userDefinedKnownSolutions.bC`.

The eigenfunctions for the heat equation on annulus of inner radius a and outer radius b take the form

$$u(\mathbf{x},t) = a\, e^{-\mathcal{D}\lambda^2 t}\Big(c_J J_n(\lambda r) + c_Y Y_n(\lambda r)\Big)\,\sin(n\theta), \tag{9}$$

for $n = 0, 1, 2, 3, \ldots$, where $\lambda = \lambda_{m,n}$, $m = 1, 2, 3, \ldots$ is the m'th root of

$$J_n(\lambda a)Y_n(\lambda b) - J_n(\lambda b)Y_n(\lambda a) = 0, \qquad \text{for Dirichlet BCs at } r = a,\ r = b, \tag{10}$$

$$J_n'(\lambda a)Y_n'(\lambda b) - J_n'(\lambda b)Y_n'(\lambda a) = 0, \qquad \text{for Neumann BCs at } r = a,\ r = b, \tag{11}$$

and where

$$c_J = \alpha\, Y_n(\lambda a), \quad c_Y = \alpha\, J_n(\lambda a), \qquad \text{for Dirichlet BCs at } r = a,\ r = b, \tag{12}$$

$$c_J = \alpha\, Y_n'(\lambda a), \quad c_Y = \alpha\, J_n'(\lambda a), \qquad \text{for Neumann BCs at } r = a,\ r = b, \tag{13}$$

for any scale factor α. The roots are computed in `cgad/codes/annulusEigenvalues.maple` and saved in some files.

Here are some results from grid convergence studies. See `cg/ad/conv/Makefile` for commands to generate these tables.

CN+FD2:

grid	N	u	r
annulus2	1	2.9e-5	
annulus4	2	7.8e-6	3.72
annulus8	4	2.0e-6	3.94
annulus16	8	5.0e-7	3.98
rate		1.96	

Table 8: Cgad, annulusEigen, method=im, Max norm, method=im, known=annulusEigen, ts=im, order=2, bdfOrder=2, dtMax=.05, $t = .5$, cfl=0.9, kappa=.1, m=1, n=0 kx=1, ky=1, kz=1, tz=poly degreex=2 degreet=2 fx=0.5 fy=0.5 fz=0.5 -ft=0.5 Thu Apr 22 07:59:08 2021

BD4+FD4:

grid	N	u	r
annulus2	1	2.4e-4	
annulus4	2	7.9e-6	30.79
annulus8	4	3.5e-7	22.33
annulus16	8	1.9e-8	18.82
annulus32	16	1.1e-9	17.31
rate		4.43	

Table 9: Cgad, annulusEigen, method=bdf4, Max norm, method=bdf4, known=annulusEigen, ts=bdf, order=4, bdfOrder=4. $t = .5$, cfl=0.9, kappa=.1, m=1, n=0 kx=1, ky=1, kz=1, tz=poly degreex=2 degreet=2 fx=0.5 fy=0.5 fz=0.5 -ft=0.5 Thu Apr 22 07:05:46 2021

5 Notes on the CHAMP interface conditions

Files in CgAd related to implementation of the CHAMP interface conditions.

- champBoundaryConditions.bC - fill CHAMP interface equations into the coefficient matrix.
- getChampResidual.bC - compute the Champ residual.
- champResidualOpt.bf90 - optimized routines for computing the residual in the CHAMP conditions.
- champMacros.h - holds some common champ bpp macros.
- implicit.C - calls champBoundaryConditions
- applyBoundaryConditions.bC : function applyBoundaryConditionsForImplicitTimeStepping() - assigns RHS for the Champ conditions.

References

[1] D. L. Brown, G. S. Chesshire, W. D. Henshaw, and D. J. Quinlan, *Overture: An object oriented software system for solving partial differential equations in serial and parallel environments*, in Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, 1997, p. .

[2] D. L. Brown, W. D. Henshaw, and D. J. Quinlan, *Overture: An object oriented framework for solving partial differential equations*, in Scientific Computing in Object-Oriented Parallel Environments, Springer Lecture Notes in Computer Science, 1343, 1997, pp. 177–194.

[3] W. D. Henshaw, *Overture: An object-oriented system for solving PDEs in moving geometries on overlapping grids*, in First AFOSR Conference on Dynamic Motion CFD, June 1996, L. Sakell and D. D. Knight, eds., 1996, pp. 281–290.