ACADIA5.0 Users Manual

Wendy C. Gentleman
May 29, 1998

Report NML-98-6

1 Introduction

ACADIA is a FORTRAN finite element formulation of the non-conservative form of the vertically integrated Advection-Diffusion-Reaction (ADR) equation that tracks NTV different transport variables (C_m, m=1...NTV) in $NLAY$ (n=1...NLAY)different fluid layers:

$$H_n \frac{\partial C_m}{\partial t} = -H_n \mathbf{V_n} \cdot \nabla C_m - C_m \sigma_n - \mathcal{Z}_m + \nabla \cdot (\mathcal{K}_n H_n \nabla C_m) + R_m(\mathbf{C}) H_n$$

where,

$C_m(x, y, t)$ is the instantaneous average concentration (number per volume) of transport variable m in the fluid layer n
$\mathbf{C}$ is array $(C_1, C_2, ..., C_{NTV})$ of transport variable concentrations
$H_n(x, y, t)$ is the thickness of the fluid layer n at location (x, y)
$\mathbf{V_n}(x, y, t)$ is the fluid velocity in layer n, with Cartesian components (u_l, v_l)
$\sigma(x, y, t)$ is the fluid source in layer n
$\mathcal{K}_n(x, y, t)$ is the horizontal diffusion coefficient for layer n
$R_m(\mathbf{C})$ is the reaction term for transport variable C_m
$\mathcal{Z}_m$ is the net upward vertical flux of C_m through the layer n (due to both fluid and behavioural motions)

Using the Galerkin weighted residual method, the solution is discretized by representing it on linear triangular elements. The integrations are performed with nodal quadrature. The time derivative is treated explicitly with forward Euler. To determine the solution, the code requires the user to define: nodal descriptions of the physical transport parameters $\mathbf{V_n}, H_n, \mathcal{K}_n, \sigma_n$ for each layer, the net vertical flux $\mathcal{Z}_m$, a reaction rate R_m, and initial and boundary conditions.

2 Software Overview

The format of the code is similar to that of QUODDY and earlier ACADIA series in that it consists of: (1) an include file containing dimensioning parameters; (2) a run-specific input file; (3) a main program; (4) a set of core subroutines that are used by the main program; (5) a set of user subroutines that are specifically adapted to each run.

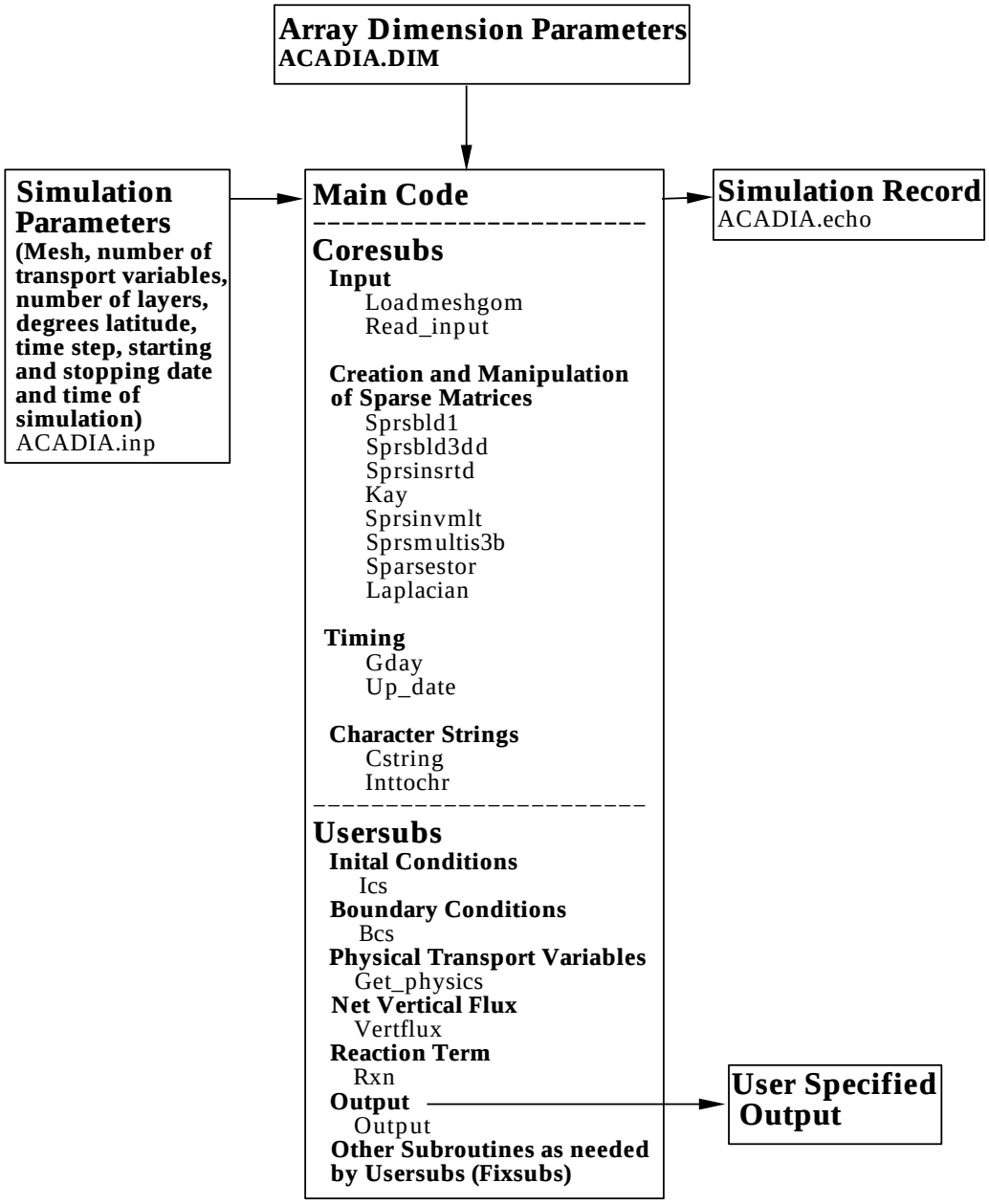

Figure 1: Structure of ACADIA5.0 code

2.1 Include file (ACADIA.DIM):

This file is used by the main program and several subroutines to dimension the arrays at compile time. Thus, the values listed here must be at least as large as those needed for both the mesh related arrays and the concentration based arrays. The parameters that must be specified are:

NNDIM – Maximum number of horizontal nodes

NEDIM – Maximum number of horizontal 2-D triangles

NTVDIM – Maximum number of transport variables.

NLAYDIM – Maximum number of transport variables.

NFTRDIM – Maximum sparsity factor (maximum elemental connectivities plus 1)

Example 'ACADIA.DIM' file for 3 transport variables in two layers on the marmap1 mesh:

```
INTEGER NNDIM, NEDIM, NFTRDIM, NTVDIM, NLAYDIM
PARAMETER (NNDIM=9000,NEDIM=17000,NFTRDIM=20,NTVDIM=3, NLAYDIM=2)
```

2.2 Input File (ACADIA.inp):

The input file is required for each run and provides the basic simulation parameters. It must conform to the ACADIA.inp standard format as described below.

Line 1 - should read the character string 'Comment:' which is the label for the next line and is ignored during data input.

Line 2 - inputs a comment of maximum 72 characters about the current simulation

Line 3 - should read 'Case name', which is the label for the next line and is ignored during data input.

Line 4 - specifies the case name (including the directory location if it is not soft-linked to the current directory). The code will append the suffixes '.nod', '.ele', and '.bat' to this string in order to know where to find the NML standard files.

Line 5 - should read 'Simulation Parameters', which is the label for the next line and is ignored during data input.

Line 6 - inputs the number of transport variables

Line 7 - inputs the number of fluid layers

Line 8 - inputs the degrees latitude of the center of the mesh

Line 9 - inputs the time step in seconds.

Line 10 - inputs the starting date (day, month, year), time (in seconds after midnight). The value of time can be greater than 8.64E4 (equivalent of 1 day) as the code will automatically adjust the day and time accordingly.

Last Line - inputs the quitting criteria for the code as day, month year and time in seconds after midnight. Again the value of time can be greater than 8.64E4. This allows the user to specify an overall length of the simulation instead of the exact date and time of the end of the simulation.

Example ACADIA.inp file for a 60 day simulation of three passive tracers in two different fluid layers on the marmap1 domain:

```
{Comment:}
ACADIA input file for 60 day passive tracers
{Case name:}
/usr/envir/wendy/meshes/marmap1/marmap1
{Simulation parameters:}
3                [number of transport variables]
2                [number of fluid layers]
43.500           [degree latitude]
360.             [time step (seconds)]
01 03 1996 0.0  [starting date (d m y) and time (sec)]
01 03 1996 5.184E6   [end date (d m y) and time (sec)]
```

2.3 Main Program:

The main program consists of the main code and the core subroutines. It is responsible for performing all of the finite element assembly, solution operations and time integration. An overview of the general structure is as follows:

It calls subroutine **READ_INPUT** to read in the basic simulation parameters from the input file. It then begins initialization. Mesh variables are loaded by the core subroutine **LOADMESHGOM**. Elemental areas and the sparse matrix versions of the basis function gradients for the mesh are computed in the core subroutine **SPARSESTOR**. Physical transport parameters $\mathbf{V}, H, \sigma/H$ and the laplacian $HAGPGP$, are set in by the user subroutine **GET_PHYSICS**. User subroutine **ICS** is called to initialize the concentration fields $C(m,i)$ for all transport variables and to define $IVARLAY$, the parameter relating the indices of the transport variables and the fluid layers. After an initialization call to subroutine **OUTPUT**, it then begins the time-stepping loop. At each time step:

i) The nodal values of the net vertical flux through the layer, $VFLUX(m,i)$, for each transport variable are evaluated in subroutine **VERTFLUX** based on the current values of the transport variables.

ii) The nodal values of the reaction rate, $RC(m,i)$, for each transport variable are evaluated in subroutine **RXN** based on the current values of the transport variables.

iii) Spatial derivatives of the concentration of each transport variable are found for each node with the core subroutine **SPRSMLTIS3b**

iv) The net local rate of change, $RHSC(m,i)$, in nodal concentration values of each transport variable (after advection, diffusion and reaction) is computed.

v) After looping through all the horizontal nodes, the boundary conditions are applied in user subroutine **BCS**.

vi) Variables are updated: timing parameters $(ITER, KD, SSEC)$, concentrations of all transport variables $C(m,i)$ and physical transport variables.

vii) Subroutine **OUTPUT** is called with the new values

viii) The code then checks the current time against the quitting criteria and then either returns for another time step or stops.

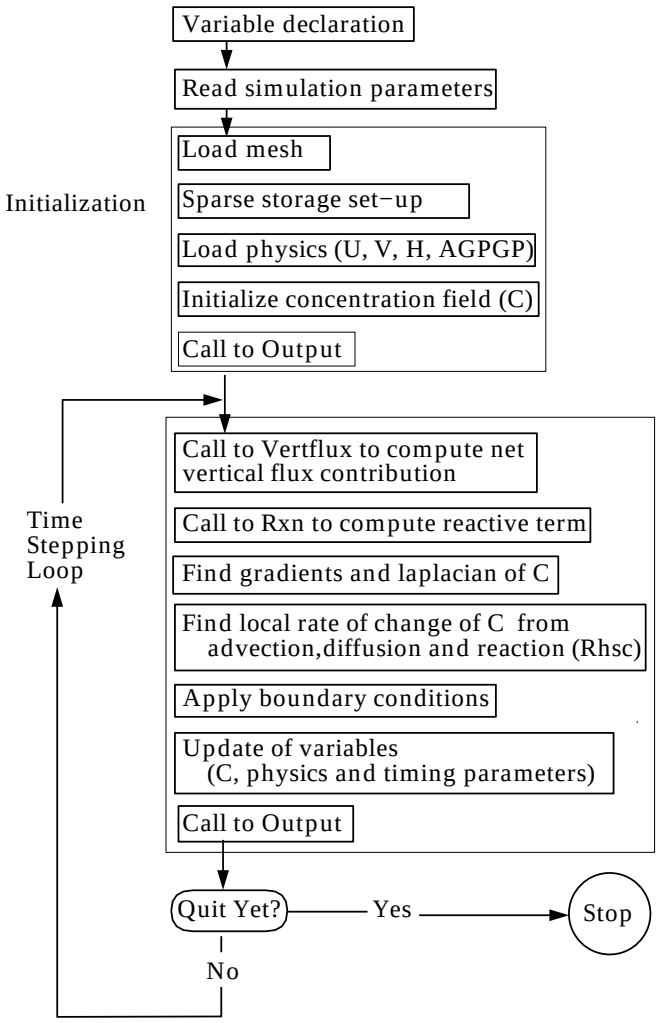

Figure 2: Flow diagram of ACADIA core program

2.4 Core Subroutines:

These are the subroutines called by the main code that perform the basic solution operations and are responsible for loading mesh files and the ACADIA.inp file. They must be linked to the main code at compile time. All ACADIA5.0 core subroutines are identical to fixed subroutines used in the QUODDY series with the exception of these four:

i) **SPRSMLTIS3b**: A sparse matrix multiplier for the new multi-transport variable array format. It differs from **SPRSMLTIS3**, used in ACADIA4.0 only in the dimensioning of $HAGPGP$.

ii) **READ_INPUT**: Reads in the ACADIA.inp file and opens the ACADIA.echo file.

iii) **SPARSESTOR**: Given the mesh information, this subroutine is responsible for assembling the sparse matrices of the gradient operators (PPX and PPY)

iv) **LAPLACIAN** computes the laplacian operator ($HAGPGP$).

2.5 User Subroutine Requirements:

There are six run-specific user subroutines that are called from the main program:

```
i) ICS (nn,ntv,casnam,
     &               x,y,kd,ssec,iter,sv,bathy,h,c,nlay,ivarlay):
```

This subroutine is called during the initialization stage of the main program in order to define to the variables: C_m and $IVARLAY$. The initial concentration field $C(m,i)$ for all NTV transport variables and all NN nodes in the domain must be prescribed. The code requires the units of $C(m,i)$ to be $No./m^3$. The variable $IVARLAY$ is the array of dimension $NTVDIM$ which tells the code which fluid layer each transport variable is in by dictating the index of the fluid layer corresponding to each variable. This must be set for all NTV varaibles. The subroutine is also passed other arguments which may be required by the user in order to construct the initial concentration field.

```
ii)  OUTPUT(nn,ntv,ne,nlay,casnam,kd,ssec,iter,delt,
     &                x,y,in,sv,ppx,ppy,iq,jq,
     &     u,v,h,ivarlay,bathy,c)
```

This subroutine is called at initialization and then once every time step with the latest values of the simulation variables. No calculation is required by the main code.

```
iii) GET_PHYSICS(casnam,nn,ne,nftr,nlay,deg,kd,ssec,
     &                      iter,x,y,in,iq,jq,sv,
     &    bathy,u,v,sigma,h,hagpgp,div)
```

This subroutine is used to specify the nodal values of the physical transport parameters U, V, H, σ for each layer at that particular time step. The user can call the core subroutine **LAPLACIAN** to compute $HAGPGP$ if $\mathcal{K}$ or H has changed since the last time step. The subroutine is also passed other arguments which may be required by the user in order to construct the velocity and diffusivity fields.

```
iv) VERTFLUX(nn,ntv,nlay,iter,kd,ssec,x,y,u,v,h,ivarlay,sv,
     &     ppx,ppy,iq,jq,bathy,c,div)
```

This subroutine is used to compute the net upward vertical flux, $VFLUX(m,i)$, through the fluid layer. The code requires the units of this variable to be $No./m^3s$. As with the other subroutines, several other arguments are passed in case they are needed by the user for the computation.

```
v) RXN(nn,ntv,nlay,iter,kd,ssec,x,y,u,v,h,ivarlay,sv,
     &     ppx,ppy,iq,jq,bathy,c,rc)
```

This subroutine is used to specify the reaction term (representing local generation or decay) for each transport variable. This is stored in the variable $RC(m,i)$ which has units of $No./m^3s$. The subroutine is passed other arguments which may be required by the user in order to construct the reaction term.

```
vi) BCs(ntv,nlay,casnam,x,y,bathy,
     &               u,v,h,ivarlay,delt,sv,iter,kd,ssec,c,rhsc)
```

At each time step, the local rate of change from advection, diffusion and reaction is computed for each transport variable and stored in $RHSC(m,i)$. Prior to integration (by multiplication with $DELT$), subroutine **BCS** is called to make adjustments to $RHSC(m,i)$ for the inclusion of the boundary integral contribution.

$$\frac{3}{A^E H_i} \oint \mathcal{K} H \frac{\partial C_m}{\partial n} \phi_i dS$$

As the main code has no knowledge of the boundary elements, the entire array of $RHSC(m,i)$ is sent. If $RHSC$ is not changed, the default boundary condition of no diffusive efflux will be applied. Should the user wish to apply other boundary conditions, the NML standard '.bel' mesh file will be required. Type 1 boundary conditions may be applied at a node by zeroing out $RHSC(m,i)$ and setting $C(m,i)$ to the value desired at the end of this time step. The subroutine is passed other arguments which may be required by the user in order to construct more complicated boundary conditions.

3 Further Considerations:

1) The main ACADIA program produces an echo file at runtime listing information about the simulation. Writing to unit=2 in the user subroutines will send data to this file.

2) Because the time integration is an explicit calculation, for the numerical model to run stably certain paramters must be taken into account. Defining:

Δx = length scale for an element
Δt = time step of model
$\frac{\Delta H}{H}$ = maximum elemental ratio of change in bathymetry to bathymetric depth
U = velocity scale (depends upon field specified by the user)
w_m= vertical behavioural velocity scale(depends upon value specified by the user),

$\mathcal{K}$=diffusivity scale (depends upon field specified by the user)
τ=time constant associated with the reaction term (depends upon value specified by the user),

a scaling analysis of the terms in the equation reveals the following considerations:

(i) Richardson/Fourier #: $\frac{\mathcal{K}\Delta t}{\Delta x^2} < 1$

(ii) Courant #: $\frac{U\Delta t}{\Delta x} - \frac{\frac{\mathcal{K}\Delta t}{\Delta x^2}\Delta H}{H} < 1$

(Generally it is a good idea to have Courant # $\simeq 0.1$)

(iii) Peclet #: $\frac{U\Delta x}{K} - \frac{\Delta H}{H} < 1$

(iv) $\Delta t < \tau$.

(iv) $\Delta t < \frac{H}{w_m}$.

4 Example Application 1: Passive Tracers

http://www-nml.dartmouth.edu/Software/acadia5.0/examples/passive/

In this example application the JF lagrangian residual circulation field is used to transport three independent passive tracers. The first one is transported in the vertically averaged flow. The other two are assumed to remain in the top 25m surface layer regardless of the fluid velocities. Initial conditions of concentration of 1.0 on the bank (defined by the 100m isobath and the 69^oW longitude) and 0.0 elsewhere are prescribed for both the passive tracer in the vertically averaged flow and one of those in the surface flow. The third passive tracer is initialized with an empty domain. The simulation is performed on the marmap1 mesh for 60 days beginning on midnight January 1, 1996 and uses a Δt of 360 seconds. The user subroutines are structured as follows:

Subroutine **ICS** determines the the onbank/offbank location of each node and assigns C_1, C_2 the appropriate concentration. C_3 is initialized to zero.

Boundary conditions are specified according to whether or not the boundary segment represents an inflow (concentration is specified: 0.0 for C_1 and C_2, 1.0 for C_3) or an outflow (a diffusive normal-flux of 0.0 is specified for all three variables). The **BCS** subroutine is set up to determine the the inflow/outflow nature of the boundary segment and then apply the correct type of condition. This subroutine requires the mesh '.bel' file.

The **OUTPUT** subroutine is set up to print out the initial conditions and simulation results every 30 days in an NML '.s2r3' file.

As the concentrate is constrained to remain within the layer, the **VERTFLUX** does nothing.

As these are passive tracers, subroutine **RXN** does nothing.

The initialization call to subroutine **GET_PHYSICS**, the JF lagrangian residual '.v2r' file, horizontal diffusivity '.s2r' file and the depth file (fluid thickness over which the velocities were averaged) are loaded. The depth and horizontal diffusivity parameters are then sent to the **LAPLACIAN** subroutine to compute **HAGPGP**. The divergence field $\frac{1}{H}\nabla(H\mathbf{V})$ is read in for the top 25m layer fluid source. Since the physical parameters are constant throughout the run, nothing is computed during subsequent calls to the subroutine.

When compiled and linked with the archived Acadia main code and fixed subroutines and run with the ACADIA.inp file provided, you should be able to generate the output files.

5 Example Application 2: Zooplankton Population Dynamics

http://www-nml.dartmouth.edu/Software/acadia5.0/examples/active/

In this example application the climatological lagrangian residual circulation field is used to transport 3 coupled variables representing generation zero (diapausers, adult males and adult females) of *Calanus finmarchicus* in the Gulf of Maine. Diapausers are carried in the vertically averaged flow (owing to their assumed deeper distribution) and adults are carried in the top 25m surface flow. The diapausing animals develop to adulthood at a fixed rate of $4\%/day$ and with a M:F sex-ratio of 50:50. Where the (bathymetry$> 200m$), the diapauser concentration is nudged towards the data with a time constant of $\alpha = .1days$. Mortality of each stage is set at a constant fraction of that stage and increased 5-fold in shoal areas (bathymetry$< 100m$). The simulation is performed on the marmap1 mesh for 60 days beginning on midnight January 1, 1996 and uses a Δt of 360 seconds. The physical fields are linearly interpolated. The user subroutines are structured as follows.

Subroutine **ICS** initializes each variable with the DJ MARMAP data: diapausers are the total of stages CIII-CV, adults are assumed to have an initial M:F sex-ratio of 10:90. This subroutine requires the archived OA files of the MARMAP data.

The **OUTPUT** subroutine is set up to print out the concentrations of each stage in an NML standard '.s2r3' file at 30 day intervals.

For all stages, boundary conditions are specified according to whether or not the boundary segment represents an inflow (a concentration of 0.0 is specified) or an outflow (a diffusive normal-flux of 0.0 is specified). The **BCS** subroutine is set up to determine the the inflow/outflow nature of the boundary segment and then apply the correct type of condition. This subroutine requires the mesh '.bel' file.

Subroutine **RXN** is used to determine the local change in concentration of each stage due to the population dynamics. The diapausers concentrations decay due to their development and mortality and increase in deep areas because of the nudging (which represents the influx of bottom sources). The adults experience an increased concentration from the developing C5s (adjusted to account for the different layer thicknesses; newly molted adults are all assumed to be in the surface layer) and a decrease from their mortality.

In the initialization call to subroutine **GET_PHYSICS**, the 3 relevant bi-monthly fields (ND, JF, MA) for velocities, horizontal mixing coefficients and, for the top25m layer, the divergence fields, are loaded. Thelayer thickness is held constant throughout the simulation.

When compiled and linked with the archived Acadia core code and fixed subroutines, and run with the ACADIA.inp file provided, you should be able to generate the output files .

Appendix A: Finite Element Formulation and Implementation of the 2-D Non-Conservative ADR Equation

The Galerkin finite element formulation of the 2-D non-conservative form of the ADR equation for the m-th transport variable for a given fluid layer (the n subscript has been dropped):

$$H\frac{\partial C_m}{\partial t} = -H\mathbf{V}\cdot\nabla C_m - C_m\sigma - \mathcal{Z} + \nabla\cdot(\mathcal{K}H\nabla C_m) + R_m(\mathbf{C})H \tag{1}$$

is the weighted residual statement:

$$\begin{aligned}\sum_{Elements} < H\frac{\partial C_m}{\partial t}\phi_i >^E \;=\; & - < H\mathbf{V}\cdot\nabla C_m\phi_i >^E - < C_m\sigma\phi_i >^E - < \mathcal{Z}\phi_i >^E \\ & + < \nabla\cdot(\mathcal{K}H\nabla C_m)\phi_i >^E + < R(\mathbf{C})\phi_i >^E \end{aligned} \tag{2}$$

where ϕ_i are the basis functions that define C_m:

$$C_m = \sum C_{m_j}\phi_j; \tag{3}$$

$$\nabla C_m = \sum C_{m_j}\nabla\phi_j; \tag{4}$$

For situations where both the horizontal boundaries of the domain are time-independent, the first term can be re-written as:

$$< H\frac{\partial C_m}{\partial t}\phi_i >^E = \sum_j \frac{\partial C_{m_j}}{\partial t} < H\phi_j\phi_i >^E \tag{5}$$

The advective flux term becomes:

$$\begin{aligned} < H\mathbf{V}\cdot\nabla C_m\phi_i >^E \;&=\; \sum_j C_{m_j} < H\mathbf{V}\cdot\nabla\phi_j\phi_i >^E \\ &=\; \sum_j C_{m_j} < Hu\frac{\partial\phi_j}{\partial x}\phi_i >^E + \sum_j C_{m_j} < Hv\frac{\partial\phi_j}{\partial y}\phi_i >^E \end{aligned} \tag{6}$$

The diffusive flux term can be re-written:

$$< \nabla\cdot(\mathcal{K}H\nabla C_m)\phi_i >^E = < \nabla\cdot(\mathcal{K}H\nabla C_m\phi_i) >^E - < \mathcal{K}H\nabla C_m\cdot\nabla\phi_i >^E \tag{7}$$

By the divergence theorem, the first term on the right hand side of the equation will only have a contribution if the element is on the boundary of the domain. Thus, summation over all the elements will convert this into a boundary integral.

$$\begin{aligned} &= \oint \mathcal{K}H\nabla C_m\cdot\hat{\mathbf{n}}\phi_i dS - \sum_{Elements} < \mathcal{K}H\nabla C_m\cdot\nabla\phi_i >^E \\ &= \oint \mathcal{K}H\frac{\partial C_m}{\partial n}\phi_i dS - \sum_{Elements}\sum_j C_{m_j} < \mathcal{K}H\nabla\phi_j\cdot\nabla\phi_i >^E \\ &= \oint \mathcal{K}H\frac{\partial C_m}{\partial n}\phi_i dS - \sum_{Elements}\left\{\sum_j C_{m_j} < \mathcal{K}H\frac{\partial\phi_j}{\partial x}\frac{\partial\phi_i}{\partial x} >^E - \sum_j C_{m_j} < \mathcal{K}H\frac{\partial\phi_j}{\partial y}\frac{\partial\phi_i}{\partial y} >^E\right\} \end{aligned} \tag{8}$$

where $\oint$ is the line integral around the boundary of the mesh.

We choose:

i) linear basis functions and triangular elements

ii) the time derivative, $\frac{\partial()}{\partial t}$, to be evaluated using the Euler explicit expression: $\frac{()^{t+1}-()^t}{\Delta t}$

iii) evaluation of the inner product, $<>^E$ through nodal quadrature:

$$< () >^E = \frac{A^E}{3}\sum_{k=1}^{3}()_k \tag{9}$$

The terms become:

$$\sum_j \frac{\partial C_{m_j}}{\partial t} < H\phi_j\phi_i >^E = \sum_j \frac{\partial C_{m_j}}{\partial t}\frac{A^E}{3}\sum_{k=1}^{3}(H\phi_j\phi_i)_k \tag{10}$$

$$= \frac{\partial C_{m_i}}{\partial t}\frac{A^E H_i}{3} \tag{11}$$

$$\sum_j C_{m_j} < Hu\frac{\partial \phi_j}{\partial x}\phi_i >^E = \sum_j C_{m_j}\frac{\partial \phi_j}{\partial x} < Hu\phi_i >^E$$

$$= \sum_j C_{m_j}\frac{\partial \phi_j}{\partial x}\frac{A^E}{3}\sum_{k=1}^{3}(Hu\phi_i)_k$$

$$= \sum_j C_{m_j}\frac{\Delta y_j (Hu)_i}{6} \tag{12}$$

$$\sum_j C_{m_j} < Hv\frac{\partial \phi_j}{\partial y}\phi_i >^E = \sum_j C_{m_j}\frac{\partial \phi_j}{\partial y} < Hv\phi_i >^E$$

$$= \sum_j C_{m_j}\frac{\partial \phi_j}{\partial y}\frac{A^E}{3}\sum_{k=1}^{3}(Hv\phi_i)_k$$

$$= -\sum_j C_{m_j}\frac{\Delta x_j (Hv)_i}{6} \tag{13}$$

$$\sum_j C_{m_j} < \sigma\phi_j\phi_i >^E = sum_j C_{m_j}\frac{A^E}{3}\sum_{k=1}^{3}(\sigma\phi_j\phi_i)_k$$

$$= sum_j C_{m_j}\frac{A^E\sigma_i}{3} \tag{14}$$

$$< \mathcal{Z}_m\phi_i >^E = \frac{A^E}{3}\sum_{k=1}^{3}(\mathcal{Z}_m\phi_i)_k$$

$$= \frac{A^E}{3}\mathcal{Z}_{m_i} \tag{15}$$

$$\sum_j C_{m_j} < \mathcal{K}H \frac{\partial \phi_j}{\partial x}\frac{\partial \phi_i}{\partial x} >^E = \sum_j C_{m_j} \frac{\partial \phi_j}{\partial x}\frac{\partial \phi_i}{\partial x} < \mathcal{K}H >^E$$

$$= \sum_j C_{m_j} \frac{\Delta y_j}{2A^E}\frac{\Delta y_i}{2A^E}\frac{A^E}{3}\sum_{k=1}^{3}(\mathcal{K}H)_k$$

$$= \sum_j C_{m_j} \frac{\Delta y_j \Delta y_i[(\mathcal{K}H)_1 + (\mathcal{K}H)_2 + (\mathcal{K}H)_3]}{12A^E}$$

which, can be written

$$= \frac{\sum_j C_{m_j}\Delta y_j \Delta y_i (\mathcal{K}H)^E_{av}}{4A^E} \tag{16}$$

where

$$(\mathcal{K}H)^E_{av} = \frac{[(\mathcal{K}H)_1 + (\mathcal{K}H)_2 + (\mathcal{K}H)_3]}{3} \tag{17}$$

Similarly

$$\sum_j C_{m_j} < \mathcal{K}H \frac{\partial \phi_j}{\partial y}\frac{\partial \phi_i}{\partial y} >^E = \sum_j C_{m_j} \frac{\partial \phi_j}{\partial y}\frac{\partial \phi_i}{\partial y} < \mathcal{K}H >^E$$

$$= \sum_j C_{m_j} (-\frac{\Delta x_j)}{2A^E}(-\frac{\Delta x_i)}{2A^E}\frac{A^E}{3}\sum_{k=1}^{3}(\mathcal{K}H)_k$$

$$= \sum_j C_{m_j} \frac{\Delta x_j \Delta x_i[(\mathcal{K}H)_1 + (\mathcal{K}H)_2 + (\mathcal{K}H)_3]}{12A^E}$$

$$= \sum_j C_{m_j} \frac{\Delta x_j \Delta x_i (\mathcal{K}H)^E_{av}}{4A^E} \tag{18}$$

$$< HR_m(\mathbf{C})\phi_i >^E = \frac{A^E}{3}\sum_{k=1}^{3}(HR_m(\mathbf{C})\phi_i)_k \tag{19}$$

$$= \frac{A^E H_i}{3} R_{m_i}(\mathbf{C}) \tag{20}$$

Putting all this together, the Galerkin weighted residual equation becomes:

$$\sum_{Elements}\sum_i \frac{(C^{k+1}_{m_i} - C^k_{m_i})}{\Delta t}\frac{A^E H_i}{3} = \sum_{Elements}\sum_i \{$$

$$\sum_j C^k_{m_j}\left[-\frac{\Delta y_j (Hu)_i}{6} + \frac{\Delta x_j (Hv)_i}{6} - \frac{(\Delta y_j \Delta y_i + \Delta x_j \Delta x_i)(\mathcal{K}H)^E_{av}}{4A^E}\right]$$

$$-\frac{A^E}{3}\sigma_i - \frac{A^E}{3}\mathcal{Z}_{m_i} + \frac{A^E H_i}{3}R_{m_i}(\mathbf{C})\Big\} + \oint \mathcal{K}H\frac{\partial C_m}{\partial n}\phi_i dS \tag{21}$$

For the terms associated with the physical transport (advection and diffusion), the summation over j over all the elements is handled by the sparse matrix routines. The spatial derivative operators for a node i are summed over all j in the elements containing i and stored in the sparse matrices $PPX, PPY, HAGPGP$. Division by the mass matrix ($SV = \frac{A^E}{3}$) is done with inverse sparse matrix multiplication. From Eqn(21), H_i drops out of the advective terms, PPX and PPY are divided

by SV, whereas $HAGPGP$ must be divided by the product $SV * H$. For each node i, the derivatives of the concentration field for each transport variable ($DCDX(m), DCDY(m), DIFFUS(m)$) are then calculated with the sparse multiplication routines. Eqn (21) also indicates that for the fluid source, vertical flux and reaction terms, division my the mass matrix can be done a priori, leaving the variable σ and the subroutines **VERTFLUX** and **RXN** to definte the terms $\frac{\sigma(n,i)}{H_i}$,$VFLUX(m,i)/H_i$ and $R(m,i)$ respectively.

Thus the concentration of the transport variable $\mathbf{C}_{m_i}$ at the new time can then be written explicitly as:

$$\begin{aligned} C_{m_i}^{k+1} &= C_{m_i}^{k} + \Delta t * [-DCDX_{m_i} u_i - DCDY_{m_i} V_i - C_{m_i}^{k} * sigmah_i - vflux_{m_i} \\ &\quad + DIFFUS_{m_i} + R_{m_i} + \text{boundary integral contribution}] \end{aligned} \tag{22}$$

In the code, the term in sqaure brackets on the right hand side is what is stored in the variable $RHSC(m,i)$

Appendix B: ACADIA Timing Parameters

There are two types of timing parameters in the code. An itteration number $ITER$ keeps track of the number of time steps taken. $ITER = 0$ at the beginning of the simulation. To avoid precision problems for longer runs or for coordination of time-domain inputs, an absolute value of time is also tracked. This method of denoting time uses two variables: an integer, KD, and a real variable $SSEC$. The integer KD is the day number according to the Gregorian calendar. By designating som reference (specifically, KD=1 on January 1, 0000), this parameter uniquely spcifies the current date. $SSEC$ is the time elapsed in seconds after midnight on KD. Thus $SSEC$ will vary between 0.0 and 8.64E4 (=24 hours * 3600 seconds/hour). $SSEC$ is incremented by $DELT$ every time step. When the value of $SSEC$ becomes larger than 8.64E4, subroutine **UP_DATE** will increment KD and reset $SSEC$ accordingly. The subroutine **GDAY** is capable of evaluating KD given a Gregorian calendar date, and conversely, evaluating the current calendar date given KD. Thus, with knowledge of these two numbers, it is possible to determine the current calendar date anywhere within the code.

Sample Call to Subroutine GDAY

Integer day,month,year

for January 7th 1996 —> day=07, month=01, year=1996

call Gday(day,month,year,kd0)

return value is kd0=729031

Sample call to Entry DMY (Part of Subroutine GDAY)

kd0=729031

call DMY(day, month,year,kd0)

return values are day=07, month=01, year=1996

Sample call to subroutine Up_Date

kd0=729031

for 11:56 PM —> ssec0 = (23*3600)+(56*60)= 86160

deltat=5.25 minutes=(5.25*60)=315

when current time gets incremented by the time step

ssec0=ssec0+deltat=86475

call up_date(kd0,ssec0)

return values are kd0=729032, ssec0=75