Flipping Indices
(Storing 3 time levels)

FORTRAN 90 code pieces:

```
…
Real, Dimension (Mx,3) : : F          !  Mx is number of grid points, defined in a prior parameter statement
…
NM1 = 1          ! mnemonic for every 'n-1' time level value of 'F' used in a finite difference formula
N = 2            ! mnemonic for every time level 'n' of 'F' used
NP1 = 3          ! mnemonic for every time level 'n+1' of 'F' used
…
! define initial condition of F    (at time t=0)
! Note: depending upon the boundary conditions, the range of the loop may be different and there may
! be additional statements to complete the time step. This example might apply for Dirichlet B.C.
Do k = 1, Mx
F(k,NM1) = ….    ! right side = initial condition formula or initial condition was read in from a file, etc.
End Do
…
! calculate result of the first time step using a forward difference (at time t = dt)
! Note: depending upon the boundary conditions, the range of the loop may be different and there may
! be additional statements to complete the time step. This example might apply for Dirichlet B.C.
First: Do k = 1, Mx
F(k,N) = …  ! right hand side is finite difference formula using F at NM1, e.g. F(k,NM1) might be 1st term
End do first
….
! Main time loop (using leap frog) for the remaining time steps
Timeloop: do j = 2, NTS   ! NTS is number of time steps. First time step done in loop 'first'

! Note: depending upon the boundary conditions, the range of the loop may be different and there may
! be additional statements to complete the time step. This example might apply for Dirichlet B.C.
Grid: Do k = 1, Mx
F(k,NP1) = F(k,NM1) + ….    ! where the other terms involve F at N, e.g. F(k+1,N) might be there…
End do grid
! flip indices so since data presently at the 'n+1' time level will be the 'n' time level the next time 'j' is
! increased. Similarly, data now at the 'n' time level will be the 'n-1' level the next time j is incremented.
! Since data at what is presently the 'n-1' time level is not needed the next time j is incremented, that
! location in storage is available to hold the 'n+1' values the next time j is increased.
Nsave = NM1
NM1 = N
N = NP1
NP1 = Nsave
End do Timeloop
…
```

Flipping Indices

(Storing 2 time levels – if numerical scheme allows*)

FORTRAN 90 code pieces:

```
…
Real, Dimension (Mx,2) : : F          !  Mx is number of grid points, defined in a prior parameter statement
…
N = 1                 ! mnemonic for every time level ‘n’ of ‘F’ used
NP1 = 2               ! mnemonic for every time level ‘n+1’ of ‘F’ and  time level ‘n-1’ locations
…
! define initial condition of F    (at time t=0)
! Note: depending upon the boundary conditions, the range of the loop may be different and there may
! be additional statements to complete the time step. This example might apply for Dirichlet B.C.
Do k = 1, Mx
F(k,NP1) = ….    ! right side = initial condition formula or initial condition was read in from a file, etc.
End Do
…
! calculate result of the first time step using a forward difference (at time t = dt)
! Note: depending upon the boundary conditions, the range of the loop may be different and there may
! be additional statements to complete the time step. This example might apply for Dirichlet B.C.
First: Do k = 1, Mx
F(k,N) = …  ! right hand side is finite difference formula using F at NP1, e.g. F(k,NP1) might be 1st term
End do first
….
! Main time loop (using leap frog) for the remaining time steps
Timeloop: do j = 2, NTS   ! NTS is number of time steps. First time step done in loop ‘first’
! Note: depending upon the boundary conditions, the range of the loop may be different and there may
! be additional statements to complete the time step. This example might apply for Dirichlet B.C.
Grid: Do k = 1, Mx
F(k,NP1) = F(k,NP1) + ….    ! where the other terms involve F at N, e.g. F(k+1,N) might be there…
End do grid
! flip indices so since data presently at the ‘n+1’ time level will be the ‘n’ time level the next time ‘j’ is
! increased. Similarly, data now at the ‘n’ time level will be the ‘n-1’ level the next time j is incremented.
! Since data at what is presently the ‘n-1’ time level is not needed the next time j is incremented, that
! location in storage is available to hold the ‘n+1’ values the next time j is increased.
! *This works as long as the values needed at the ‘n-1’ time level are at the current grid point location ‘k’
! in loop ‘Grid’ or are at a location (>k) that is yet to be calculated as one cycles through loop ‘Grid’.
! Often, can’t do this but need 3 times levels if have a diffusion term or are doing time filtering.
!
Nsave = N
N = NP1
NP1 = Nsave
End do Timeloop
```