1 Implicit multi-step methods

The third-order Adams Moulton method uses the time-levels $t_{n+1}$, $t_n$, and $t_{n-1}$ to interpolate the right hand side function $f$ to third-order accuracy using quadratic polynomials. Derive the coefficients $a_i$ of this third-order integration method:

$$u^{n+1} = u^n + \Delta t \left( a_{-1} f^{n+1} + a_0 f^n + a_1 f^{n-1} \right)$$

(1)

Note that a modified version of this algorithm uses a predictor-corrector approach where the predicted value is obtained from a third-order explicit scheme and used in the evaluation of the right hand side term $f^{n+1}$.

2 Leap-Frog Trapezoidal time stepping

One remedy to the computational mode of the Leap-Frog scheme is to combine it with the trapezoidal step. The combined scheme can be written as follows:

$$u^* = u^{n-1} + 2\Delta t f(u^n, t_n)$$

(2)

$$u^{n+1} = u^n + \Delta t f \left( \frac{u^n + u^*}{2}, t_{n+\frac{1}{2}} \right)$$

(3)

The scheme can be viewed as a predictor corrector method. The first step predicts $u$ at time level $n + 1$ using a leap-frog scheme, while the second one is a trapezoidal step centered at $n + \frac{1}{2}$. Unlike the pure trapezoidal scheme, this one is explicit and uses the average of $u^n$ and $u^*$ as an estimate of $u^{n+\frac{1}{2}}$. Investigate the stability of the scheme for the case $f = \kappa u$ by deriving the expression for the amplification factor. Is there a parasitic mode? and if yes which one? Plot the amplification factor for $\kappa \Delta t = z = x + iy$ where $|x| < 1$ and $0 \leq y \leq 2$. Compare its stability region to that of the Leap-Frog, RK2 and AB2 schemes.

Plotting stability diagrams is rather cumbersome especially if they involve the square roots of complex numbers. Here is a sample matlab script to achieve that.

```matlab
M=201; xmin =-1; xmax=1.; dx=(xmax-xmin)/(M-1); x =xmin:dx:xmax;
N=201; ymin = 0; ymax=2.; dy=(ymax-ymin)/(N-1); y =ymin:dy:ymax;
z= x'*ones(size(y)) + i * ones(size(x))'*y;
% roots of equation a A^2 + b*A +c, where a,b and c are functions of z
% a=;
% b=;
% c=;
det = sqrt(b.^2-4*a.*c);
A1 = (-b+det)./(2*a); A2 = (-b-det)./(2*a);
A = max(abs(A1),abs(A2));
contour(x,y,A',[1 1]); % will draw the contour of neutral stability
```
3 Linear shallow water equations

The linearized non-dimensional shallow water equations in a channel of unit depth and unit width are

\[ u_t + \eta_x = 0 \]  \hspace{1cm} (4)  
\[ v_t + \eta_y = 0 \]  \hspace{1cm} (5)  
\[ \eta_t + u_x + v_y = 0 \]  \hspace{1cm} (6)  

\((u, v, \eta)\) are the velocity, in the \((x, y)\) direction and the hydrostatic pressure due to the motion of the free surface. The solution in space can be described by a Fourier series whose expansion takes the following form:

\[ u(x, y, t) = \hat{u}(t) \sin px \cos qy \]  \hspace{1cm} (7)  
\[ v(x, y, t) = \hat{v}(t) \cos px \sin qy \]  \hspace{1cm} (8)  
\[ \eta(x, y, t) = \hat{\eta}(t) \cos px \cos qy \]  \hspace{1cm} (9)  

where \((p, q)\) are wavenumbers in the \((x, y)\) directions such that \(k^2 = p^2 + q^2\).

1. Write down the time-ODEs governing the evolution of the Fourier coefficients \(\hat{u}, \hat{v}\), and \(\hat{\eta}\) as a system of the form

\[ \frac{d\vec{w}}{dt} + P\vec{w} = 0 \]  \hspace{1cm} (10)  

2. Show that the Fourier amplitude should satisfy the following energy equation:

\[ \left(\frac{\dot{\hat{u}}^2 + \dot{\hat{v}}^2}{2} + \frac{\dot{\hat{\eta}}^2}{2}\right)_t = 0 \]  \hspace{1cm} (11)  

3. Show that the eigenvalue/eigenvector pairs of this system are

\[ \lambda_1 = -i\sqrt{p^2 + q^2}, \quad \lambda_2 = i\sqrt{p^2 + q^2}, \quad \lambda_3 = 0, \quad V = \begin{pmatrix} p & -p & -q \\ q & -q & p \\ i\sqrt{p^2 + q^2} & -i\sqrt{p^2 + q^2} & 0 \end{pmatrix} \]  \hspace{1cm} (12)  

4. Verify that

\[ V^{-1} = \frac{1}{2(p^2 + q^2)} \begin{pmatrix} p & q & -i\sqrt{p^2 + q^2} \\ -p & -q & -i\sqrt{p^2 + q^2} \\ -2q & 2p & 0 \end{pmatrix} \]  and \(V^{-1}PV = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \]  \hspace{1cm} (13)  

5. Use the above result to define the characteristic variables of the system \(\vec{\omega}' = V^{-1}\vec{\omega}\).  

6. Based on the eigenvalues of this system, which ODE integration schemes are suitable for the numerical solution.
4 Time Stepping Schemes

Write a program to integrate equations 10 using a TVDRK3 (Gottlieb and Shu, 1998), RK4, and AB3 schemes. The initial conditions are \( \hat{u} = \hat{v} = 0 \) and \( \hat{\eta} = 1 \). Use \( p = 4\pi \) and \( q = 3\pi \). Integrate until time \( T = 40 \) using \( \Delta t = 0.2, 0.1, 0.05, 0.02, 0.01, \) and \( 0.005 \). Monitor the evolution of \( \frac{(u^2 + v^2 + \eta^2)}{2} \).

The exact solution for the problem is:

\[
\hat{u}_e(t) = \frac{p}{\sigma} \sin \sigma t, \quad \hat{v}_e(t) = \frac{q}{\sigma} \sin \sigma t, \quad \hat{\eta}_e(t) = \cos \sigma t \quad \text{where} \quad \sigma^2 = p^2 + q^2. \tag{14}
\]

Hence the wave period is \( T = \frac{2\pi}{\sqrt{p^2 + q^2}} = 2/5 \). Compare the numerical solution with the exact solution at \( t = 40 \), that is after 400 cycles. Plot the error metric:

\[
\epsilon = \sqrt{(\hat{u} - \hat{u}_e)^2 + (\hat{v} - \hat{v}_e)^2 + (\hat{\eta} - \hat{\eta}_e)^2}
\]

at \( t = 40 \) as a function of \( \Delta t \) using a log-log scale and estimate the slope of the convergence curve. Explain how you decide on choosing the time-step. Use only the stable time-steps to estimate the slope and discard the other ones from your plots.

4.1 TVD-RK3

The TVD-RK3 scheme is a third order Runge-Kutta time-stepping scheme with 3 stages. Its intermediate stages can be seen as successive corrections to \( u^{n+1} \) using a convex combination of weights. In brief the scheme can be described algorithmically as follows:

\[
\begin{align*}
\hat{u}^{(1)} &= u^n + \Delta t f(u^n) \tag{15} \\
\hat{u}^{(2)} &= \frac{3}{4} u^n + \frac{1}{4} \hat{u}^{(1)} + \frac{1}{4} \Delta t f(\hat{u}^{(1)}) \tag{16} \\
u^{n+1} &= \frac{1}{3} u^n + \frac{2}{3} \hat{u}^{(2)} + \frac{2}{3} \Delta t f(\hat{u}^{(2)}) \tag{17}
\end{align*}
\]

Notice that in this scheme only a single temporary value and a single right hand side vector \( f \) need to be stored at any point of the stage. Notice also that the original value, \( u^n \) can only be over-written at the end of the third step.

4.2 RK4

The RK4-scheme can be summarized below as follows:

\[
\begin{align*}
\hat{u}^{(1)} &= u^n + \frac{\Delta t}{2} f(u^n) \tag{18} \\
\hat{u}^{(2)} &= u^n + \frac{\Delta t}{2} f(\hat{u}^{(1)}) \tag{19} \\
\hat{u}^{(3)} &= u^n + \Delta t f(\hat{u}^{(2)}) \tag{20} \\
u^{n+1} &= u^n + \Delta t \left( \frac{f(u^n) + 2f(\hat{u}^{(1)}) + 2f(\hat{u}^{(2)}) + f(\hat{u}^{(3)})}{6} \right) \tag{21}
\end{align*}
\]
This scheme is a little more complicated than the first one, and at first glance, seems to require a lot more storage. A closer examination reveals that this is not true: we still need to store the intermediate variables, and the corresponding right hand side; in addition we need to store the accumulation of the sum that appears in the last step, before we actually overwrite the array $f$.

### 4.3 AB3

This is a 4 time-level scheme that has the form:

$$u^{n+1} = u^n + \Delta t \frac{23f(u^n) - 16f(u^{n-1}) + 5f(u^{n-2})}{12}$$

We need to use a start-up procedure for this scheme, a good candidate is the RK3 scheme as both are of the same order. This scheme does not require a temporary value but does require the storage of old right hand sides $f$’s, at least three of them. It also seems to require a lot of copying but again a little closer examination shows that this is not true. All we have to do is trick a subroutine call. This is illustrated in 3.

### 4.4 Programming Considerations

The program should be divided into section. The first one initializes the variables and sets the numerical and physical parameters (like $p$, $q$, $\Delta t$ and the number of time steps). The second part consists of the time-loop in which you call a subroutine to advance the solution from time level $n$ to time level $n + 1$. Within the loop you should design an RK4 subroutine that takes care of the four stages of the RK4 time step. Each of these stages requires its own evaluation of the right hand side $-Pu$. A good program would then isolate the computations of the right hand side in a separate subroutine. Finally, and besides the time-stepping, the time-loop should call a diagnostics routine that writes out quantities of interest like the kinetic energy or error measures.

The program becomes easier to write by breaking it into functional modules. Here we have broken it into an initialization part, a time-loop, a time-stepping routine (RK4), a routine that computes the right hand side of the ODE (RHS), and a diagnostics routine. Sections of codes can now be re-used or changed easily. For example changing the initial conditions requires modifying the subroutine Initialize, changing the time-stepping routine involve substituting another algorithm for RK4 (an RK3 or AB3 for example), and finally if the equations change, all we have to do is substitute the appropriate RHS routine. In pseudo-code the program should look as shown in the attached pages. Notice that we have used modules to keep routines that are similar in functionalities within the same file. The use of modules is highly recommended as it helps develop clear and robust codes that are easy to test.
program ode
  use timeint ! use time-integration module
  use rhsmod, only : Diagnostics ! use for diagnostics
  implicit none
  integer, parameter :: nn=3 ! dimension of dependent variable array
  integer, parameter :: ndiag=5 ! print diagnostics every ndiag steps
  real(r8) :: q(nn) ! q(1) is u, q(2) is v, and q(3) is p
  real(r8) :: dt ! timestep
  real(r8) :: ntimestep ! number of timesteps to take

  write(6,*)'Enter dt, ntimestep'
  read(5,*) dt, ntimestep ! read parameters
  call Initialize(q,nn) ! initialize q
  do n = 1,ntimestep ! time-loop
    time = (n-1)*dt
    call RK3(q,dt,time,nn) ! do a single RK3 step
    ! call RK4(q,dt,time,nn) ! do a single RK4 step
    ! call AB3(q,dt,time,nn) ! do a single AB3 step
    if (mod(n,isnap)==0) then
      time = n*dt
      call Diagnostics(q,err,KE,time,nn) ! get Diagnostics
      call History(q,time,nn) ! Save solution history
    endif
  enddo
stop
end program ode

Figure 1: Main time-integrator program that coordinates the initialization, time-stepping, and post-processing
module modrks
    use rhsmod
    contains
    subroutine RK4(q,dt,time,nn)
        implicit none
        integer, intent(in) :: nn
        real(r8), intent(in) :: dt,time
        real(r8), intent(inout) :: q(nn) ! on input q is at time n, and on output at time n+1
        real(r8) :: r(nn)
        call RHS(r,q,time,nn) ! calculate the right hand side
        return
    end subroutine RK4
    subroutine RK3(q,dt,time,nn)
        implicit none
        integer, intent(in) :: nn
        real(r8), intent(in) :: dt,time
        real(r8), intent(inout) :: q(nn) ! on input q is at time n, and on output at time n+1
        return
    end subroutine RK3
end module modrks

Figure 2: Module for two-time levels multistage methods.
module modab3
use modrhs
use modrk3
contains
subroutine AB3(q, dt, time, nn)
  integer, isave :: imod=-2  ! keep track of time step
  select case (imod)
    case (-2)
      call RHS(q, r2, nn)
      call RK3(q, dt, nn)  ! start-up first RK3 step
      imod = -1  ! indicator for 2nd start-up
    case (-1)
      call RHS(q, r1, nn)
      call RK3(q, dt, nn)  ! start-up 2nd RK3 step
      imod = 0  ! indicator to switch to regular AB3
    case (0)
      call AB3Work(q, r0, r1, r2, dt, nn); imod = 1
    case (1)
      call AB3Work(q, r2, r0, r1, dt, nn); imod = 2
    case default
      call AB3Work(q, r1, r0, r2, dt, nn); imod = 0
  end select
return
end subroutine AB3

subroutine AB3Work(q, r0, r1, r2, dt, nn)
  real(r8), intent(in) :: r2(nn)  ! rhs at time n-2
  real(r8), intent(in) :: r1(nn)  ! rhs at time n-1
  real(r8), intent(inout) :: r0(nn)  ! rhs at time n
  real(r8), intent(inout) :: q(nn)  ! solution at time n/n+1 on input/output
  real(r8), parameter :: a0= 23_r8/12_r8,a1=-16_r8/12_r8,a2= 5_r8/12_r8
  call RHS(q, r0, nn)
  q = q + dt*(a0*r0+a1*r1+a2*r2)
end subroutine AB3Work
end module modab3

Figure 3: AB3 algorithm that avoids copying of old right hand side values
module rhsmod
implicit none
contains
subroutine RHS(r,q,time,nn) ! compute right hand side
  integer, intent(in) :: nn
  real(r8), intent(in) :: time
  real(r8), intent(in) :: q(nn) ! function values needed to compute rhs
  real(r8), intent(in) :: r(nn) ! right hand sides
  return
end subroutine RHS
subroutine Diagnostics(q,err,KE,time,nn) ! diagnostics
  return
end subroutine Diagnostics
subroutine Initialize(q,nn) ! Set the initial conditions
  return
end subroutine Initialize
end module rhsmod