STUDY OF SPACECRAFT ORBITS IN THE GRAVITY FIELD OF THE MOON

-ANNEXES-

by

EDGAR CARDOSO VILANA

A dissertation submitted to the Department of Aerospace Engineering,
ETSEIAT – Universitat Politècnica de Catalunya,
in partial fulfillment of the requirements for the degree of
Aeornautical Engineer

Tutor: Dr. Elena Fantino

January 2012

ESCOLA TÈCNICA SUPERIOR D'ENGINYERIES INDUSTRIAL I AERONÀUTICA DE TERRASSA

ENGINYERIA SUPERIOR AERONÀUTICA

Escola Tècnica Superior d'Enginyeries Industrial i Aeronàutica de Terrassa

UNIVERSITAT POLITÈCNICA DE CATALUNYA

This page is intentionally left blank.

TABLE OF CONTENTS

ANNEX A.

FORTRAN90 SCRIPTS

A.1. Orbit_Integration.f90

```
!****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!****************************************************************
!Main of the program
!****************************************************************

program Orbit_Integration

      use InputOutput
      use Gravity
      use CoordTransformC2S
      use Kepler
      use NumericalIntegration

      implicit none

      ! Local variables
      real(pr)                      :: r,phi,lambda,U0
      real(pr)                      :: x,y,z,s,t,u

      real(pr), dimension(3)        :: dVs_P,dVc

      character*1                   :: key
      real*8                        :: time_beg,time_end

      integer                       :: i,m,aux

      !New variables
      real(pr), dimension(3)         :: acc
      real(pr), dimension(6)         :: pos_vel, e, field
      real(pr), dimension(6)         :: b, f
      real(pr), dimension(13,6)      :: array
      real(pr), dimension(6)         :: intr_var
      real(pr)                       :: time, tf, counter,
short_step, step2
      real(pr)                       :: r_mod, orbit_period,
long_step
      logical(pr)                    :: control, first, overcome
      real(pr)                       ::
acceleration,distance2,central_grav
```

```
      time = 0.
      acc = 0.
      aux = 0
      counter = 0.
      step2 = 0.
      control = .true.
      overcome = .true.

      ! Determine starting time
      call cpu_time(time_beg)

      call WriteToLogFile('   Time of start =
','(f14.4)',time_beg)

      ! Computation of basic angles and conversion factors
      call Greeks

      ! Read programme specifications from namelist input
      call ReadInputParameters

      call WriteToLogFile('   Input parameters read','',0.d0)

      ! Check consistency of input parameters (GRAVITY FIELD)
      call CheckConsistency

      call WriteToLogFile('   Consistency of input parameters
verified','',0.d0)

      !************* ALLOCATE ARRAYS:*****************************

      ! Stokes coefficients
      call Stokes_AllocDealloc(.true.,N_min,N_max,M_min,M_max)

      ! rm, im

      call RmIm_AllocDealloc(.true.,M_min,M_max, 1)

      call cosphi2m_AllocDealloc(.true.,M_min,M_max)

      ! Am, Bm
      call LumpedCoeff_AllocDealloc(M_min,M_max,.true.)

      ! Helmholtz polynomials and their derivatives up to order
three
      call Helmholtz_AllocDealloc(.true.,N_min,N_max,M_min,M_max)

      ! HP normalization coefficients
      call HPCoeff_AllocDealloc(.true.,N_min,N_max,M_min,M_max)

      ! rho_n
      call rho_AllocDealloc(.true.,N_min,N_max)

      ! arrays containing output gravity gradients
      call GravityGradients_AllocDealloc(.true.)

      call WriteToLogFile('   Arrays allocated','',0.d0)
```

```
      call
ReadStokesCoeff(12,MoonModelFile,N_min,N_max,M_min,M_max)

      !*********************************************************

      !** INITIAL ORBITAL ELEMENTS & POSITION_VELOCITY VECTOR*****

    if(orb_el) then
        e(1) = sm_axis
        e(2) = ecc
        e(3) = incl
        e(4) = raan
        e(5) = per_arg
        e(6) = anom
        call elevec(GM_moon,e,pos_vel)
    else
        pos_vel(1) = pos_x
        pos_vel(2) = pos_y
        pos_vel(3) = pos_z
        pos_vel(4) = vel_x
        pos_vel(5) = vel_y
        pos_vel(6) = vel_z
        call vecele(GM_moon,pos_vel,e)
    endif

    r_mod =
sqrt(pos_vel(1)*pos_vel(1)+pos_vel(2)*pos_vel(2)+pos_vel(3)*pos_ve
l(3))

    orbit_period = (twopi * sqrt(e(1)*e(1)*e(1)/GM_moon))

    if(orbits) then
        tf = num_orbits * orbit_period      ! number of orbits
    else
        tf = num_days * 24 * 3600           ! number of days
    endif

    short_step = orbit_period/ppo
    long_step = tf/orbits_step

      !************************ NUMERICAL INTEGRATION************

      call WriteOutputFiles(aux, time, pos_vel, e)

      do while((time .LT. tf) .AND. (r_mod .GT. (ae_moon+9.e3)))

          call
RungeKutta78(time,pos_vel,6,h,h_min,h_max,e1,array,b,f,acc)

          call vecele(GM_moon,pos_vel,e)

          r_mod =
sqrt(pos_vel(1)*pos_vel(1)+pos_vel(2)*pos_vel(2)+pos_vel(3)*pos_ve
l(3))

          if(overcome) then
              if(r_mod .GT. (ae_moon+2000.e3)) then
                  print *, '   Beyond low-altitude orbit
at',time,'s'
```

```
                overcome = .false.
            endif
        endif

        if(time .LE. orbit_period) then

            if(time .GT. counter) then
                call WriteOutputFiles(aux, time, pos_vel, e)
                print *, time
                counter = counter + short_step
            endif

        endif

      if(control) then
            step2 = step2 + long_step
            control = .false.
            first = .true.
        endif

        if(time .GT. step2 + orbit_period) control = .true.

        if(time .GT. orbit_period) then

            if(time .GT. step2) then

                if(first) then
                    counter = time
                    first = .false.
                endif

                if(time .GT. counter) then
                    call WriteOutputFiles(aux, time, pos_vel, e)
                    print *, time
                    counter = counter + short_step
                endif
            endif

        endif

    end do

    aux = 2

    if(r_mod .LT. (ae_moon+11.e3)) print *, '   Collide with the
lunar surface at',time,'s'

    call WriteOutputFiles(aux, time, pos_vel, e)


      !**************** Deallocate arrays***********************

      ! Stokes coefficients
      call Stokes_AllocDealloc(.false.,N_min,N_max,M_min,M_max)

      ! rm, im
      call RmIm_AllocDealloc(.false.,M_min,M_max, 1)

      call cosphi2m_AllocDealloc(.false.,M_min,M_max)
```

```
      ! Am, Bm
      call LumpedCoeff_AllocDealloc(M_min,M_max,.false.)

      ! Helmholtz polynomials and their derivatives up to order
three
      call Helmholtz_AllocDealloc(.false.,N_min,N_max,M_min,M_max)

      ! HP normalization coefficients
      call HPCoeff_AllocDealloc(.false.,N_min,N_max,M_min,M_max)

      ! rho_n
      call rho_AllocDealloc(.false.,N_min,N_max)

      ! arrays containing output gravity gradients
      call GravityGradients_AllocDealloc(.false.)

      call WriteToLogFile('   Arrays deallocated','',0.d0)

      ! Determine ending time
      call cpu_time(time_end)

      call WriteToLogFile('   Time of end = ','(f14.4)',time_end)
      call WriteToLogFile('   Execution time =
','(f14.4)',time_end-time_beg)

      pause 'Press any key to finish'

      !************************************************************

      stop

end program Orbit_Integration
```

A.2. Constants.f90

```
!*****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!*****************************************************************

module Constants

      implicit none

      INTEGER, PARAMETER :: pr = SELECTED_REAL_KIND (p = 14)

      real(pr), parameter :: ZERO = 0.0_pr, ONE = 1.0_pr, TWO =
2.0_pr, THREE = 3.0_pr

      real(pr), parameter :: HALF = 0.5_pr, QUART = 0.25_pr


end module Constants
```

A.3. CoordTransformC2S.f90

```
!*****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!*****************************************************************

module CoordTransformC2S

  use Constants
  use LatLongRadius

  implicit none


      contains

!*****************************************************************
      subroutine Cartesian2Spherical(ae,x,y,z,lambda,phi,r)
!*****************************************************************

      implicit none

      real(pr), intent(in)          :: ae,x,y,z
      real(pr), intent(out)         :: r,phi,lambda

      real(pr)                            ::
r2,sinphi,cosphi,sinlam,coslam

      ! geocentric distance
      r2 = x * x + y * y + z * z
      r  = sqrt(r2)

      if(r < 1.0e-30) then
            pause 'CoordTransformC2S: position vector has zero
length'
            stop
      endif
      if(r < ae) then
            pause 'CoordTransformC2S: position vector is lower
than mean Moon radius'
            stop
      endif

      sinphi = z/r
      phi = asin(sinphi)
      cosphi = cos(phi)
      if(abs(cosphi) < 1e-30) then
            pause 'CoordTransformC2S: latitude = +- pi/2'
            stop
      endif

      coslam = x / (r*cosphi)
      sinlam = y / (r*cosphi)
      lambda = atan2(sinlam,coslam)
```

```
      return
      end subroutine Cartesian2Spherical
end module CoordTransformC2S
```

A.4. Gravity.f90

```
!*****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!*****************************************************************
! Computation of geopotential and its derivatives of order 1 to 3
! according to Pines method with the lumped coefficients
implementation
!*****************************************************************

MODULE Gravity

      use LumpedCoefficients

      IMPLICIT NONE

      SAVE

      real(pr), dimension(:,:),     allocatable :: dV,d2V,d3V
      real(pr), save, private                   ::
ai(4),aij(9),aijk(20)

      PUBLIC ::   FirstDerivative, GravityGradients_AllocDealloc

      CONTAINS

!*****************************************************************
  SUBROUTINE GravityGradients_AllocDealloc(bAllocate)
!*****************************************************************

      implicit none

      ! Argument list

      logical, intent(in)           :: bAllocate

      if(bAllocate) then

            if(.not.allocated(dV))        allocate(dV(1,3))
            dV          = ZERO

            return

      endif

      if(allocated(dV)) deallocate(dV)

      return

 END SUBROUTINE GravityGradients_AllocDealloc

!*****************************************************************
  SUBROUTINE FirstDerivative(i,m_min,m_max,s,t,u,r,U1)
!*****************************************************************
```

```
!* Computes the first-order gradient at (x,y,z) from Stokes
coefficients of potential
!* of irregular shape
!*****************************************************************

    IMPLICIT NONE

      ! Argument List
      INTEGER, INTENT(IN)                 :: i,m_min,m_max
      REAL(pr), INTENT(IN)                :: s,t,u,r
    REAL(pr), DIMENSION(3), INTENT(OUT)   :: U1

      ! Local variables
    INTEGER                               :: m,m_1
      REAL(pr)                            :: rmi,imi,rm_1,im_1

      U1 = ZERO

      ai = ZERO

      do m = m_min,m_max

            rmi = rm(m,i) * cosphi2m(m)
            imi = im(m,i) * cosphi2m(m)

            ai(4) = ai(4) - Am(m,5) * rmi - Bm(m,5) * imi
            ai(3) = ai(3) + Am(m,4) * rmi + Bm(m,4) * imi

            if(m > 0) then
                  m_1  = m - 1
                  rm_1 = rm(m_1,i) * cosphi2m(m_1)
                  im_1 = im(m_1,i) * cosphi2m(m_1)
                  ai(1) = ai(1) + Am(m,2) * rm_1 + Bm(m,2) * im_1
                  ai(2) = ai(2) + Am(m,3) * rm_1 + Bm(m,3) * im_1
            endif

      enddo

      U1(1) = (ai(1) + s * ai(4)) / r
      U1(2) = (ai(2) + t * ai(4)) / r
      U1(3) = (ai(3) + u * ai(4)) / r

      return

  END SUBROUTINE FirstDerivative

!*****************************************************************

END MODULE Gravity
```

A.5. Hellenic.f90

```
!*****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!*****************************************************************
! Computation of basic angles and conversion factors
!*****************************************************************

module Hellenic

      use Constants

      implicit none

      real(pr) :: raddeg,degrad
      real(pr) :: pih, pi,twopi

      contains

      !*************************************************************
        subroutine greeks
      !*************************************************************
      ! to set some basic constants
      ! raddeg: rad to deg
      ! degrad: deg to rad
      !*************************************************************

              implicit none

              pih       = atan2(ONE,ZERO)
              pi        = pih + pih
              twopi           = pi + pi
              raddeg    = 180.0_pr/pi
              degrad    = pi/180.0_pr

              return

        end subroutine greeks

end module Hellenic
```

A.6. Helmholtz.f90

```
!*****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!*****************************************************************
! Computes Helmholts polynomials and their derivatives up to order
three
!*****************************************************************

module Helmholtz

use Constants

implicit none

SAVE

REAL(pr), PRIVATE, SAVE, dimension(:),   allocatable     ::f
REAL(pr), PRIVATE, SAVE, dimension(:,:), allocatable     ::g,h,k
real(pr), dimension (:,:),               allocatable     ::nm,dHnm

      contains

!*****************************************************************
      subroutine
Helmholtz_AllocDealloc(bAllocate,n_min,n_max,m_min,m_max)
!*****************************************************************

      implicit none

      ! Argument list
      logical, intent(in)          :: bAllocate
      integer, intent(in)          :: n_min,n_max,m_min,m_max

      if(bAllocate) then

            if(.not.allocated(Hnm))      allocate
(Hnm(0:n_max,0:n_max))
            if(.not.allocated(dHnm))     allocate
(dHnm(0:n_max,0:n_max))

            return

      endif

    if(allocated(Hnm))             deallocate (Hnm)
    if(allocated(dHnm))            deallocate (dHnm)

      return

      end subroutine Helmholtz_AllocDealloc
```

```
!*****************************************************************
      subroutine HelmholtzPolynomials_Initialize()
!*****************************************************************

      implicit none

      Hnm = ZERO
      dHnm = ZERO

      return

      end subroutine HelmholtzPolynomials_Initialize


!*****************************************************************
  SUBROUTINE HelmholtzPolynomials(n_min,n_max,m_min,m_max,x)
!*****************************************************************
!  x = sin(phi)
!*****************************************************************
    IMPLICIT NONE

    ! Argument list
    INTEGER,  INTENT(IN)      :: n_min,n_max,m_min,m_max
    REAL(pr), INTENT(IN)      :: x

    ! Local variables
    logical, save             :: bFirst=.true.
    INTEGER                   :: n, m, n_1, mp1, mp2, mp3

    if (bFirst) then
            call HelmholtzPolynomials_Initialize()
            call HP_NormalizationCoeff(n_min,n_max,m_min,m_max)
            bFirst = .false.
    endif

      Hnm(0,0)  = ONE

      ! Sectorials (m = n)
      DO n = 1, n_max

            n_1 = n-1
            Hnm(n,n)  = f(n) * Hnm(n_1,n_1)

      END DO

      ! Subsectorials
      DO n = 1, n_max

            n_1 = n-1
            Hnm(n,n_1) = g(n,n_1) * x * Hnm(n_1,n_1)

      END DO


      ! All other terms
      DO m = 0, n_max-2

            DO n = m+2, n_max
```

```
                  Hnm(n,m) = g(n,m) * x * Hnm(n-1,m)- h(n,m) *
Hnm(n-2,m)

            END DO

      END DO

    ! Derivatives of Hnm with respect to u

    do  n = 1,n_max

            do m = 0,n-1

                  mp1 = m + 1
                  mp2 = m + 2
                  mp3 = m + 3

                  dHnm(n,m) = Hnm(n,mp1) * k(n,m)

                  if(m == n-1) cycle

                  if(m == n-2) cycle

        enddo

    enddo

    return

  END SUBROUTINE HelmholtzPolynomials


!*****************************************************************
  subroutine
HPCoeff_AllocDealloc(bAllocate,n_min,n_max,m_min,m_max)

!*****************************************************************
  ! allocate arrays containing normalization coefficients
!*****************************************************************

    implicit none

      ! Argument list
      logical, intent(in) :: bAllocate
      INTEGER, intent(in) :: n_min,n_max,m_min,m_max

      integer           :: n1,n2

      if(bAllocate) then

         ! note that all coefficients must be computed, not only
those
         ! pertaining to the intervals (n_min,n_max) (m_min,m_max)
         ! because the recursions must be computed from the
beginning.

         n1 = 1
         n2 = 2
```

```
        if(.not.allocated(f))  allocate ( f(n1:n_max) )
        if(.not.allocated(g))  allocate ( g(n1:n_max,0:n_max) )
        if(.not.allocated(h))  allocate ( h(n2:n_max,0:n_max) )

        if(.not.allocated(k)) then
           allocate ( k(n1:n_max,0:n_max) )
           k = ZERO
        endif

       call InitializeHPCoeff()

     else
        if(allocated(f)) deallocate (f)
        if(allocated(g)) deallocate (g)
        if(allocated(h)) deallocate (h)
        if(allocated(k)) deallocate (k)
     endif

     return

   end subroutine HPCoeff_AllocDealloc


!*****************************************************************
   subroutine InitializeHPCoeff()
!*****************************************************************

       implicit none

       f = ZERO
       g = ZERO
       h = ZERO
       k = ZERO

       return

end subroutine InitializeHPCoeff


!*****************************************************************
   SUBROUTINE HP_NormalizationCoeff(n_min,n_max,m_min,m_max)
!*****************************************************************
!* Fills three arrays with the coefficients used in carrying out
the recursions
!* to compute the Helmholtz polynomials with Full Normalization
(according to
!* Heiskanen Moritz).
!* Proper coefficients are computed to be employed in the full
normalization of the
!* Helmholtz polynomials and their derivatives
!*****************************************************************

     IMPLICIT NONE

     ! Argument list
     INTEGER, intent(in)       :: n_min,n_max,m_min,m_max
     ! Local variable list
     INTEGER                   :: n,m,n_1,npm,n_m
     REAL(pr)                  :: two_n, sq_two
```

```
      sq_two = sqrt(TWO)

    ! f: normalization factors including coefficients from the
recurrence relation R1: Eq.(57)
    if(n_max.ge.1) then
       f(1) = sqrt(THREE)
    endif

    DO n = 1, n_max

       two_n = TWO * n

       if(n > 1) then
                  f(n) = sqrt( (two_n + ONE) / two_n )
         endif

         n_1 = n-1

         ! g,h,k
       DO m = 0, n_1

          npm = n + m
          n_m = n - m

          g(n,m) = sqrt( (two_n + ONE) * (two_n - ONE) / npm /
n_m)

          if (n > 1 .and. m < n_1)  then
             h(n,m) = g(n,m) / g(n_1,m)

          endif

          k(n,m) = sqrt( n_m * (npm + ONE))

       END DO

       k(n,0) = k(n,0) / sq_two

    END DO

    return

  END SUBROUTINE HP_NormalizationCoeff

end module Helmholtz
```

A.7. InputOutput.f90

```
!*****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!*****************************************************************
! Reads input specifications for programme execution
!*****************************************************************

module InputOutput

      use Hellenic

      implicit none

      SAVE

      integer           :: N_min,M_min,N_max,M_max
      integer           :: num_orbits,num_days,ppo,orbits_step
      real(pr)          :: ae_moon,ae_earth,a_earth,a_moon
      real(pr)          ::
GM_moon,GM_earth,GM_sun,SF,light_sp,refl,area_to_mass
      real(pr)          :: height,dlam,dphi,lam0,phi0,lamf,phif
      logical           ::
orb_el,orbits,PinesGrav,Earth3rdbody,Sun3rdbody
      logical           :: SunPressure,Thermal
      character*150     ::
MoonModelFile,OrbitFile,OrbitPath,OutputPath,RecordData
      character*6       ::
dbl_fmt,qdp_fmt,dec_fmt,exp_fmt,output_fmt
      character*6       :: MoonModel
      character*4       :: Code
      integer           :: nGGFiles
      real(pr)          :: sm_axis,ecc,incl,per_arg,raan,anom
      real(pr)          :: pos_x,pos_y,pos_z,vel_x,vel_y,vel_z
      real(pr)          :: h,h_min,h_max,e1
      integer           :: aux1,aux2,aux3,aux4

      contains

      !*************************************************************
      subroutine ReadInputParameters
      !*************************************************************
      !  to read namelist /input/

      !*************************************************************

            implicit none

            real*8      :: dae_moon,dae_earth,da_earth,da_moon
            real*8      :: dGM_moon,dGM_earth,dGM_sun

namelist/nm_accelerations/Earth3rdbody,Sun3rdbody,SunPressure,Ther
mal
namelist/nm_field_limits/N_min,M_min,N_max,M_max
```

```
namelist/nm_intr_variables/sm_axis,ecc,incl,per_arg,raan,anom,orb_
el
namelist/nm_position_velocity/pos_x,pos_y,pos_z,vel_x,vel_y,vel_z
namelist/nm_integration_time/orbits,num_orbits,num_days,ppo,orbits
_step
namelist/nm_integration_parameters/h,h_min,h_max,e1
namelist/nm_SRP_parameters/SF,light_sp,refl,area_to_mass
namelist/nm_parameters/dae_moon,dae_earth,da_earth,da_moon,dGM_moo
n,dGM_earth,dGM_sun
namelist/nm_model_file/MoonModelFile,MoonModel
namelist/nm_output/OutputPath

            ! read input from namelist
            open(7,file='C:\Users\Edgar\Desktop\PFC\Programa
Pines. E.
FANTINO\Orbit_Integration_Project\Orbit_Integration\InputFiles\inp
ut_Orbit_Integration.txt')

            read(7,nml = nm_accelerations)
            read(7,nml = nm_field_limits)
            read(7,nml = nm_intr_variables)
            read(7,nml = nm_position_velocity)
            read(7,nml = nm_integration_time)
            read(7,nml = nm_integration_parameters)
            read(7,nml = nm_SRP_parameters)
            read(7,nml = nm_parameters)
            read(7,nml = nm_model_file)
            read(7,nml = nm_output)

            close(7,status='keep')

            ae_moon = dae_moon
            ae_earth = dae_earth
            a_earth = da_earth
            a_moon = da_moon
            GM_moon = dGM_moon
            GM_earth = dGM_earth
            GM_sun = dGM_sun

            incl = incl * degrad
            per_arg = per_arg * degrad
            raan = raan * degrad
            anom = anom * degrad

            aux1 = 0
            aux2 = 0
            aux3 = 0
            aux4 = 0

            if(Earth3rdbody .EQ. .true.) aux1 = 1
            if(Sun3rdbody .EQ. .true.) aux2 = 1
            if(SunPressure .EQ. .true.) aux3 = 1
            if(Thermal .EQ. .true.) aux4 = 1

            code = 'PiLC'

            ! Precision == 8
                  write(output_fmt,'(a6)') dbl_fmt
            ! Precision == 16
```

```
            !     write(output_fmt,'(a6)') qdp_fmt

            return

      end subroutine ReadInputParameters

!*****************************************************************
      subroutine CheckConsistency
!*****************************************************************

            implicit none

            ! check on degree and order limits

            if((M_min > M_max) .or. (N_min > N_max) &
                  .or. (N_min < 0) .or. (M_min < 0) .or. (N_max <
0) .or. (M_max < 0)) then
                  pause 'Inconsistency on degree/order limits'
                  stop
            endif

            if(M_max > N_max) then
                  pause 'M_max > N_max will be set to N_max'
                  M_max = N_max
            endif

            return

      end subroutine CheckConsistency

!*****************************************************************
      subroutine WriteToLogFile(string,fmt,value)
!*****************************************************************

            implicit none
            character*(*), intent(in)          :: string
            character*(*), intent(in)          :: fmt
            real*8, intent(in)                 :: value

            if(len(fmt) == 0) then
                  write(14,'(a)') trim(string)
                  write(6,'(a)') trim(string)
            else
                  write(14,'(a$)') trim(string)
                  write(14,trim(fmt)) value
                  write(6,'(a$)') trim(string)
                  write(6,trim(fmt)) value
            endif

            return

      end subroutine WriteToLogFile

!*****************************************************************
      subroutine WriteOutputFiles(i, time, pos_vel, e)
!*****************************************************************

    implicit none
```

```
    integer i
    real(pr), dimension(6)       :: pos_vel, e
    real(pr)                     :: time

    if(i .EQ. 0) then

        open (unit=5,file=trim(OutputPath)//'time.txt',
action='write')
          open (unit=6,file=trim(OutputPath)//'orb_elements1.txt',
action='write')
          open (unit=7,file=trim(OutputPath)//'orb_elements2.txt',
action='write')
          open (unit=8,file=trim(OutputPath)//'position.txt',
action='write')
          open (unit=9,file=trim(OutputPath)//'velocity.txt',
action='write')

    endif

    if(i .EQ. 2) then

        close(5)
        close(6)
        close(7)
        close(8)
        close(9)

    endif

    if(i .NE. 2) then

          write(5,*) time
        write(6,*) e(1), e(2), e(3)
        write(7,*) e(4), e(5), e(6)
        write(8,*) pos_vel(1), pos_vel(2), pos_vel(3)
        write(9,*) pos_vel(4), pos_vel(5), pos_vel(6)

    endif

    i = 1

    end subroutine WriteOutputFiles

end module InputOutput
```

A.8. Kepler.f90

```
!*****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!*****************************************************************

module Kepler

use Constants
use Hellenic

contains

!*****************************************************************
   subroutine elevec(xmu,e,s)
!*****************************************************************
!  From orbital elements e to state vector s (r(i), v(i))
!
!  Input:
!  xmu = gravitational parameter (GM) of central body, any units
!  e(i) = vector of 6 elements:
!  a = e(1)   semi-major axis (same units as given by xmu)
!  e = e(2)   eccentricity
!  i = e(3)   inclination, in interval 0 to pi
!  o = e(4)   ascending node, in interval 0 to twopi
!  w = e(5)   arg. of pericenter, in interval 0 to twopi
!  v = e(6)   true anomaly, in interval 0 to twopi
!  output:  state vector s(i) of 6 elements (position and
velocity):
!  r(1),r(2),r(3) = s(1),s(2),s(3)
!  v(1),v(2),v(3) = s(4),s(5),s(6)
!*****************************************************************

    implicit none

    ! Arguments
    double precision, intent(in)    :: xmu
    double precision, intent(in)    :: e(6)
    double precision, intent(out)   :: s(6)

    ! Locals
    double precision                :: p,f,cv,ecv,r,u
    double precision                ::
cu,su,co,so,ci,si,cocu,sosu,cosu,socu
    double precision                :: fx,fy,fz,vr,vu

    p = e(1)*(1.d0 - e(2)**2)
    ! safety measure for the square root
    p = dmax1(p,1.d-30)
    f = sqrt(xmu/p)
    cv = cos(e(6))
    ecv = 1.d0 + e(2)*cv
    r = p/ecv
    u = e(5) + e(6)
    cu = cos(u)
```

```
    su = sin(u)
    co = cos(e(4))
    so = sin(e(4))
    ci = cos(e(3))
    si = sin(e(3))
    cocu = co*cu
    sosu = so*su
    socu = so*cu
    cosu = co*su
    fx = cocu - sosu*ci
    fy = socu + cosu*ci
    fz = su*si
    vr = f*e(2)*sin(e(6))
    vu = f*ecv
    s(1) = r*fx
    s(2) = r*fy
    s(3) = r*fz
    s(4) = vr*fx - vu*(cosu + socu*ci)
    s(5) = vr*fy - vu*(sosu - cocu*ci)
    s(6) = vr*fz + vu*cu*si

    return

    end subroutine elevec


!*****************************************************************
   subroutine vecele(xmu,s,e)
!*****************************************************************
!  From state vector s (r(i), v(i)) to orbital elements e
!
!  Input:
!  xmu = gravitational parameter (GM) of central body, any units
!  s(i) = state vector s(i) of 6 elements (position and velocity):
!  r(1),r(2),r(3) = s(1),s(2),s(3)
!  v(1),v(2),v(3) = s(4),s(5),s(6)
!  Output:
!  e(i) = vector of 6 elements:
!  a = e(1)   semi-major axis (same units as given by xmu)
!  e = e(2)   eccentricity
!  i = e(3)   inclination, in interval 0 to pi
!  o = e(4)   ascending node, in interval 0 to twopi (0 if i = 0)
!  w = e(5)   arg. of pericenter, in interval 0 to twopi (0 if e =
0)
!  v = e(6)   true anomaly, in interval 0 to twopi
!*****************************************************************
    implicit none

    ! Arguments
    double precision, intent(in)    :: xmu
    double precision, intent(in)    :: s(6)
    double precision, intent(out)   :: e(6)

    ! Locals
    double precision                ::
c1,c2,c3,cc,cc12,v02,r0v0,r02
    double precision                :: x,cx,ste,cte,u,c,r0
```

```
      c1=s(2)*s(6)-s(3)*s(5)
      c2=s(3)*s(4)-s(1)*s(6)
      c3=s(1)*s(5)-s(2)*s(4)
      cc12 = c1*c1+c2*c2
      cc = cc12 + c3*c3
      c = sqrt(cc)
      v02=s(4)**2+s(5)**2+s(6)**2
      r0v0=s(1)*s(4)+s(2)*s(5)+s(3)*s(6)
      r02=s(1)**2+s(2)**2+s(3)**2
      r0=sqrt(r02)
      x=r0*v02/xmu
      cx=cc/xmu
      ste=r0v0*c/(r0*xmu)
      cte=cx/r0-1.d0
      e(1)=r0/(2.d0-x)
      e(2)=sqrt(ste*ste+cte*cte)
      e(3)=atan2(sqrt(cc12),c3)
      if(cc12.gt.cc*1.d-20) goto 10
      u = atan2(s(2),s(1))*sign(1.d0,c3)
      e(4)=0.d0
      goto 20
10    u = atan2(c*s(3),s(2)*c1-s(1)*c2)
      e(4) = atan2(c1,-c2)
20    if(e(2) .gt. 1.d-20) goto 30
      e(6) = u
      e(5)=0.d0
      goto 40
30    e(6) = atan2(ste,cte)
      e(5)=u-e(6)
40    if(e(4) .lt. 0.d0) e(4) = e(4) + twopi
      if(e(5) .lt. 0.d0) e(5) = e(5) + twopi
      if(e(6) .lt. 0.d0) e(6) = e(6) + twopi

      return

      end subroutine vecele


end module kepler
```

A.9. LatLongRadius.f90

```
!*****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!*****************************************************************
! Compute various position-dependent quantities
!*****************************************************************

module LatLongRadius

      use Constants

      IMPLICIT NONE

      SAVE

      real(pr) :: cosphi,sinphi,cos2phi,cos3phi,sin2phi,sin3phi
      real(pr) :: sinlam,coslam,sin2lam,cos2lam,sin3lam,cos3lam
      real(pr) :: u2,u3,s2,s3,t2,t3,stu,st,su,tu,xu,xs,xt
      real(pr) :: GMor,GMor2,GMor3,GMor4
      real(pr) :: r2,r3

      real(pr), dimension(:), allocatable :: rhon
      real(pr), dimension(:,:), allocatable :: rm,im
      real(pr), dimension(:), allocatable :: cosphi2m

      PUBLIC ::  SinCosPhi,TrigLambda,rho_AllocDealloc

      CONTAINS

      !*************************************************************
      subroutine SinCosPhi(r,phi)
      !*************************************************************

            implicit none

            real(pr), intent(in) :: r,phi

            cosphi    = cos(phi)
            sinphi    = sin(phi)

            cos2phi   = cosphi * cosphi
            cos3phi   = cos2phi * cosphi
            sin2phi   = sinphi * sinphi
            sin3phi   = sin2phi * sinphi

            r2 = r * r
            r3 = r2 * r

            return

      end subroutine SinCosPhi

      !*************************************************************
      subroutine RmIm_AllocDealloc(bAllocate,m_min,m_max,n_points)
```

```
      !*************************************************************
            implicit none

            logical, intent(in) :: bAllocate
            integer, intent(in) :: n_points,m_min,m_max

            if(bAllocate) then
               if(.not. allocated(rm))
      allocate(rm(m_min:m_max,n_points))
               if(.not. allocated(im))
      allocate(im(m_min:m_max,n_points))

               rm = ZERO
               im = ZERO

               return

            endif

          if(allocated(rm))   deallocate(rm)
          if(allocated(im))   deallocate(im)

            return

      end subroutine RmIm_AllocDealloc

      !*************************************************************
      subroutine cosphi2m_AllocDealloc(bAllocate,m_min,m_max)
      !*************************************************************

            implicit none

            logical, intent(in) :: bAllocate
            integer, intent(in) :: m_min,m_max

            if(bAllocate) then
               if(.not. allocated(cosphi2m))
                        allocate(cosphi2m(m_min:m_max))
               cosphi2m = ZERO
               return
            endif

          if(allocated(cosphi2m))   deallocate(cosphi2m)

            return

      end subroutine cosphi2m_AllocDealloc

!*****************************************************************
      subroutine Calc_cosphi2m(phi,m_min,m_max)
!*****************************************************************

            implicit none

            real(pr), intent(in)    :: phi
            integer, intent(in)     :: m_min,m_max
            integer                 :: m
            real(pr)                :: c
```

```
            c = cos(phi)

            do m = m_min,m_max

                  if(m == 0) then
                        cosphi2m(0) = ONE
                  else
                        cosphi2m(m) = cosphi2m(m-1) * c
                  endif

            enddo

            return

      end subroutine Calc_cosphi2m

!*****************************************************************
subroutine Calc_rm_im(m_min,m_max,n_points,lambda0,dlam)
!*****************************************************************
!* Computes rm and im for m = 0, m_max and stores them in an array
!*****************************************************************
! Input:
! m_min           = min value of order m
! m_max           = max value of order m
! n_points  = number of data points over parallel
! lambda0   = geocentric longitude of first point on parallel
! dlam            = step in lambda between neighbouring points
!*****************************************************************

            implicit none

            ! Argument list
            INTEGER, INTENT(IN)     :: m_min,m_max,n_points
            REAL(PR), INTENT(IN)    :: lambda0,dlam

            ! Local variables
            integer                 :: m,i
            real(pr)                :: lambda
            real(pr)                :: s,t,rr,ii,r_1,i_1

            lambda = lambda0

            do i = 1,n_points

                  s = cos(lambda)
                  t = sin(lambda)

                  DO m = 0, m_max

                        if(m == 0) then

                              rr = ONE
                              ii = ZERO

                        else if(m == 1) then

                              rr = s
                              ii = t
```

```
                              r_1 = ONE
                              i_1 = ZERO

                        else

                              r_1 = rr
                              i_1 = ii

                              rr = s * r_1 - t * i_1
                              ii = s * i_1 + t * r_1

                        endif

                       if(m_min <= m) then

                              rm(m,i) = rr
                              im(m,i) = ii

                        endif

                    END DO

                    lambda = lambda + dlam

              enddo

              return

          end subroutine Calc_rm_im


!*****************************************************************
      subroutine rho_AllocDealloc(bAllocate,n_min,n_max)
!*****************************************************************

              implicit none

              logical, intent(in) :: bAllocate
              integer, intent(in) :: n_min,n_max

              if(bAllocate) then
                    allocate(rhon(n_min:n_max))
                    rhon = ZERO
                    return
              endif

              if(allocated(rhon)) deallocate(rhon)

              return

        end subroutine rho_AllocDealloc


!*****************************************************************
      subroutine rho2n(ae,r,GM,n_min,n_max)
!*****************************************************************

              implicit none
```

```
            ! Argument list
            real(pr), intent(in) :: ae,r,GM
            integer, intent(in)  :: n_min,n_max

            ! Local variables
            real(pr) :: rho
            integer  :: n

           rho = ae / r

            rhon(0) = GM / r

            loop_n: do n = 1,n_max

                  rhon(n) = rhon(n-1) * rho

            enddo loop_n

            return

      end subroutine rho2n

      !*************************************************************
        subroutine TrigLambda(lambda)
      !*************************************************************

        implicit none

            real(pr), intent(in) :: lambda

            sinlam = sin(lambda)
            coslam = cos(lambda)

            sin2lam = sinlam * sinlam
            cos2lam = coslam * coslam

            sin3lam = sin2lam * sinlam
            cos3lam = cos2lam * coslam

            return

        end subroutine TrigLambda

!*****************************************************************
      subroutine Calc_stu_functions(s,t,u)
!*****************************************************************

            implicit none

            real(pr), intent(in)         :: s,t,u

            s2 = s * s
            s3 = s2 * s
            t2 = t * t
            t3 = t2 * t
            stu =  s * t * u
            st = s * t
            su = s * u
            tu = t * u
```

```
            xs = ONE - s2
            xt = ONE - t2

            return

      end subroutine Calc_stu_functions


      subroutine Calc_u_functions(u)

            implicit none

            real(pr), intent(in)          :: u

            u2 = u * u
            u3 = u2 * u
            xu = ONE - u2

            return

      end subroutine Calc_u_functions

end module LatLongRadius
```

A.10. LumpedCoefficients.f90

```
!*****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!*****************************************************************
! Accumulation of lumped coefficients for derivatives of various
orders
!*****************************************************************

module LumpedCoefficients

  use Stokes
  use LatLongRadius
  use Helmholtz

  implicit none

  SAVE

  REAL(pr), SAVE, dimension(:,:), allocatable  :: Am,Bm

  CONTAINS


!*****************************************************************
      subroutine LumpedCoeff_AllocDealloc(m_min,m_max,bAllocate)
!*****************************************************************
! Allocate arrays containing lumped coefficients for computation
of
! the geopotential and various derivatives
!*****************************************************************

      implicit none

      logical, intent(in)     :: bAllocate
      integer, intent(in)     :: m_min,m_max

      integer                 :: iDim

      if(bAllocate) then

            iDim = 5

            if(.not.allocated(Am)) allocate (Am(m_min:m_max,iDim))
            if(.not.allocated(Bm)) allocate (Bm(m_min:m_max,iDim))

      else

            if(allocated(Am))       deallocate (Am)
            if(allocated(Bm))       deallocate (Bm)

      endif

      return
```

```
      end subroutine LumpedCoeff_AllocDealloc

!*****************************************************************
      subroutine InitializeLumpedCoeff
!*****************************************************************
!     Set lumped coefficients to zero
!*****************************************************************

            implicit none

            Am = ZERO
            Bm = ZERO

            return

      end subroutine InitializeLumpedCoeff

!*****************************************************************
      subroutine CalcLumpedCoeff_Der1(n_min,n_max,m,u)
!*****************************************************************
!   Computation and storage of lumped coefficients for first
derivative
!   of geopotential V
!*****************************************************************

      implicit none

      ! Argument List
      integer,  INTENT(in)    :: n_min,n_max,m
      real(pr), intent(in)    :: u

      ! Local variables
      integer                 :: n,n0,m_1
      real(pr), dimension(4)  :: add_a,add_b
      real(pr)                :: cnm,snm,cnn,r2n,H,dH,f1,L

      add_a = ZERO
      add_b = ZERO

      m_1 = m - 1
      n0 = max(m,n_min)

      do n = n0,n_max

            r2n = rhon(n)

            H = Hnm(n,m)
            dH = dHnm(n,m)

            L = H * (n + m + 1) + u * dH

            ! zonals
            if(m == 0) then

                  cnn = Clm(n,n)

                  add_a(1) = ZERO
                  add_b(1) = ZERO
                  add_a(2) = ZERO
```

```
                add_b(2) = ZERO
                add_a(3) = dH * cnn
                add_b(3) = ZERO
                add_a(4) = L * cnn
                add_b(4) = ZERO

            else

                cnm = Clm(n,m_1)
                snm = Clm(m_1,n)

                add_a(1) = H * cnm
                add_b(1) = H * snm

                add_a(2) =  add_b(1)
                add_b(2) = -add_a(1)

                add_a(3) = dH * cnm
                add_b(3) = dH * snm

                add_a(4) = L * cnm
                add_b(4) = L * snm

            endif

            f1 = r2n * m

            Am(m,2:3) = Am(m,2:3) + add_a(1:2) * f1
            Bm(m,2:3) = Bm(m,2:3) + add_b(1:2) * f1
            Am(m,4:5) = Am(m,4:5) + add_a(3:4) * r2n
            Bm(m,4:5) = Bm(m,4:5) + add_b(3:4) * r2n

        enddo

        return

        end subroutine CalcLumpedCoeff_Der1

end module LumpedCoefficients
```

A.11. OrbitalPerturbations.f90

```
!*****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!*****************************************************************
module OtherPerturb

use InputOutput

contains

!*****************************************************************
    subroutine OtherAccel(pos, ti, e_acc, s_acc, srp_acc,
thrm_acc)
!*****************************************************************

    implicit none

    real(pr), dimension(3)      :: e_acc, s_acc, srp_acc, thrm_acc
    real(pr), dimension(6)      :: pos
    real(pr), dimension(3)      :: r_sat, rse, rsm, rem
    real(pr), dimension(3)      :: r_s_sat, u_s_sat, u_sat
    real(pr)                    :: n_e, theta_e, n_m, theta_m, ti,
psr, ptr, r_sat2
    logical                     :: ecl_moon, ecl_earth

    rse = 0.
    rsm = 0.
    rem = 0.

    r_sat(1) = pos(1)
    r_sat(2) = pos(2)
    r_sat(3) = pos(3)

    ! Moon wrt the Earth

    n_m = sqrt(GM_earth/(a_moon * a_moon * a_moon))
    theta_m = n_m * ti

    rem(1) = a_moon * cos(theta_m)
    rem(2) = a_moon * sin(theta_m)

    if(Sun3rdbody .OR. SunPressure) then

        ! Earth wrt the Sun

        n_e = sqrt(GM_sun/(a_earth * a_earth * a_earth))
        theta_e = n_e * ti

        rse(1) = a_earth * cos(theta_e)

        rse(2) = a_earth * sin(theta_e)

        ! Moon wrt the Sun
```

```
        rsm = rse + rem

    endif

    if(Earth3rdbody) then
        call ThirdBody(r_sat, -rem, e_acc, GM_earth)
    endif

    if(Sun3rdbody) then
        call ThirdBody(r_sat, -rsm, s_acc, GM_sun)
    endif

    if(SunPressure) then

        srp_acc = 0.

        call Eclipse(r_sat, -rsm, ae_moon, ecl_moon)
        call Eclipse(rem+r_sat, -rse, ae_earth, ecl_earth)

        if(ecl_moon .AND. ecl_earth) then
            psr = SF/light_sp
            r_s_sat = rsm + r_sat
            u_s_sat = r_s_sat/sqrt(dot_product(r_s_sat,r_s_sat))
            srp_acc = psr * (refl) * area_to_mass * u_s_sat
        endif

    endif

    if(Thermal) then

        r_sat2 = dot_product(r_sat,r_sat)
        ptr = 977 * (ae_moon*ae_moon/r_sat2)/light_sp
        u_sat = r_sat/sqrt(r_sat2)
        thrm_acc = ptr * (refl) * area_to_mass * u_sat

    endif

    return

    end subroutine OtherAccel


!*****************************************************************
    subroutine ThirdBody(r_sat, r_body, b_acc, GM_body)
!*****************************************************************

    implicit none

    real(pr), dimension(3)  :: r_sat, r_body, rho, b_acc
    real(pr)                :: r_body2, r_body3, rho2, rho3,
GM_body

    rho = r_body - r_sat
    rho2 = rho(1)*rho(1) + rho(2)*rho(2) + rho(3)*rho(3)
    rho3 = rho2 * sqrt(rho2)

    r_body2 = r_body(1)*r_body(1) + r_body(2)*r_body(2) +
r_body(3)*r_body(3)
    r_body3 = r_body2 * sqrt(r_body2)
```

```
    b_acc = GM_body * (rho/rho3 - r_body/r_body3)

    return

    end subroutine ThirdBody

!*****************************************************************
    subroutine Eclipse(r_b_sat, r_b_s, ae_b, ecl)
!*****************************************************************

    implicit none

    ! b = body
    ! s = sun
    ! sat = satellite

    real(pr), dimension(3)  :: r_b_sat, r_b_s, u_b_s, vect_prod
    real(pr)                :: ae_b, mod_b_s
    logical                 :: ecl

    ecl = .true.

    mod_b_s = sqrt(dot_product(r_b_s,r_b_s))
    u_b_s = r_b_s / mod_b_s

    if(dot_product(r_b_sat,u_b_s) .LT. 0.) then
        vect_prod(1) = r_b_sat(2) * u_b_s(3) - r_b_sat(3) *
u_b_s(2)
        vect_prod(2) = r_b_sat(3) * u_b_s(1) - r_b_sat(1) *
u_b_s(3)
        vect_prod(3) = r_b_sat(1) * u_b_s(2) - r_b_sat(2) *
u_b_s(1)

        if(dot_product(vect_prod,vect_prod) .LT. ae_b * ae_b) then
            ecl = .false.
        endif

    endif

    return

    end subroutine Eclipse

end module OtherPerturb
```

A.12. Pines.f90

```
!*****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!*****************************************************************

module Pines

use InputOutput
use Gravity
use CoordTransformC2S

implicit none

contains

!*****************************************************************
    subroutine PinesAccel(pos_vel, acc)
!*****************************************************************

!************************ METHOD OF PINES ***********************

    implicit none

      ! Local variables
      real(pr)                :: r,r2,phi,lambda
      real(pr)                :: x,y,z,s,t,u

      real(pr), dimension(3)  :: dVc

      character*1             :: key
      real*8                  :: time_beg,time_end

      integer                 :: i,iRow,iCol,m

      real(pr), dimension(:,:), allocatable   :: pos

      real(pr), dimension(3)                  :: acc
      real(pr), dimension(6)                  :: pos_vel
      real(pr)                                :: time



        iRow = 1
        iCol = 1

      ! Extract components of cartesian Moon-fixed position vector
          x = pos_vel(1)
          y = pos_vel(2)
          z = pos_vel(3)

          r2 = x*x + y*y + z*z
          r = sqrt(r2)

      ! Determine spherical coordinates of field point
```

```
            call Cartesian2Spherical(ae_moon,x,y,z,lambda,phi,r)

                  s = x/r
                  t = y/r
                  u = z/r

      ! Compute and store various powers of ae over r
            call rho2n(ae_moon,r,GM_moon,N_min,N_max)

            call Calc_rm_im(M_min,M_max,1,lambda,0.d0)

      ! Compute trigonometric functions of phi (latitude)
            call SinCosPhi(r,phi)

            call Calc_cosphi2m(phi,M_min,M_max)

            call Calc_u_functions(u)
            call Calc_stu_functions(s,t,u)

            call Calc_cosphi2m(phi,M_min,M_max)

      ! compute fully normalized Helmholtz polynomials and their
derivatives
            call
HelmholtzPolynomials(N_min,N_max,M_min,M_max,sinphi)

      ! initialize lumped coefficients arrays
            call InitializeLumpedCoeff

            do m = M_min,M_max
                  call CalcLumpedCoeff_Der1(N_min,N_max,m,u)
            enddo

      ! if rotation of the tensors is required, then additional
trigonometric functions
      ! are to be computed
            call TrigLambda(lambda)

      ! compute first-order gradient of gravitational potential
            call FirstDerivative(1,M_min,M_max,s,t,u,r,dVc)

      ! storage in output array
            dV(iCol,:) = dVc

            acc(:) = dV(iCol,:)

end subroutine PinesAccel

end module Pines
```

A.13. RungeKutta78.f90

```
!*****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!*****************************************************************

module NumericalIntegration

use InputOutput
use VectorField

implicit none

contains

!*****************************************************************
    subroutine RungeKutta78(x,y,n,h,hmi,hmax,e1,r,b,f,acc)
!*****************************************************************
!
!  THIS ROUTINE IS AN IMPLEMENTATION OF A RUNGE-KUTTA-FEHLBERG
!  METHOD OF ORDERS 7 AND 8. USING A TOTAL OF 13 STEPS (AND
!  EVALUATIONS OF THE VECTORFIELD), IT COMPUTES TWO DIFFERENT
!  ESTIMATIONS OF THE NEXT POINT. THE DIFFERENCE BETWEEN BOTH
!  ESTIMATIONS (WITH LOCAL ERRORS OF ORDER 8 AND 9) IS COMPUTED
!  AND THE L1 NORM IS OBTAINED. THIS NORM IS DIVIDED BY N (THE
!  NUMBER OF EQUATIONS). THE NUMBER OBTAINED IN THIS WAY IS
REQUIRED
!  TO BE LESS THAN A GIVEN TOLERANCE E1 TIMES (1+0.01*DD) WHERE DD
!  IS THE L1 NORM OF THE POINT COMPUTED TO ORDER 8. IF THIS
!  REQUIREMENT IS SATISFIED THE ORDER 8 ESTIMATION IS TAKEN AS THE
!  NEXT POINT. IF NOT, A SUITABLE VALUE OF THE STEP H IS OBTAINED
!  AND THE COMPUTATION IS STARTED AGAIN.
!  IN ANY CASE, WHEN THE NEXT POINT IS COMPUTED, A PREDICTION OF
!  THE STEP H, TO BE USED IN THE NEXT CALL OF THE ROUTINE, IS
!  DONE.
!
!  in:
!  x = current value of the independent variable
!  y(i) (i = 1,...,n) = current value of the dependent variable,
!                       with:
!  n = dimension of the dependent variable (vector size)
!  h = time step to be used
!  hmi = minimum allowed value for the absolute value of h
!  hmax = maximum allowed value for the absolute value of h
!  e1 = tolerance
!  deriv = name of the routine that computes the vector field (to
be
!  declared as external in the calling module)
!
!  out:
!  x = next value of the independent variable
!  y(i) (i = 1,...,n) = the estimated next value for the dependent
!                       variable
!  h = time step to be used in the next call to this routine
!
```

```
!  Auxiliary parameters:
!
!  r = (13 * n) array to be used as working space
!  b = vector of size n to be used as working space
!  f = vector of size n to be used as working space
!!
!*****************************************************************
    implicit none

    !Arguments
    integer, intent(in)     :: n
    real(pr), intent(inout) :: x,y(n),h,r(13,n),b(n),f(n)
    real(pr), intent(in)    :: hmi,hmax,e1
    real(pr), dimension(3)  :: acc

    ! Locals:
    logical, save           :: bFirst = .true.
    real(pr), save          :: alpha(13),beta(79),c(11),cp(13)
    real(pr)                :: a,bet,d,dd,e3,e4,fac
    integer                 :: jk,j,l,k,j1

    if(bFirst) then

        ! initialization
         alpha(1) = 0.0_pr
         alpha(2) = 2.0_pr / 27.0_pr
         alpha(3) = 1.0_pr / 9.0_pr
         alpha(4) = 1.0_pr / 6.0_pr
         alpha(5) = 5.0_pr / 12.0_pr
         alpha(6) = .5_pr
         alpha(7) = 5.0_pr / 6.0_pr
         alpha(8) = 1.0_pr / 6.0_pr
         alpha(9) = 2.0_pr / 3.0_pr
         alpha(10) = 1.0_pr / 3.0_pr
         alpha(11) = 1.0_pr
         alpha(12) = 0.0_pr
         alpha(13) = 1.0_pr

         beta(1) = 0.0_pr
         beta(2) = 2.0_pr / 27.0_pr
         beta(3) = 1.0_pr / 36.0_pr
         beta(4) = 1.0_pr / 12.0_pr
         beta(5) = 1.0_pr / 24.0_pr
         beta(6) = 0.0_pr
         beta(7) = 1.0_pr / 8.0_pr
         beta(8) = 5.0_pr / 12.0_pr
         beta(9) = 0.0_pr
         beta(10) = -25.0_pr/16.0_pr
         beta(11) = -beta(10)
         beta(12) = .5e-1_pr
         beta(13) = 0.0_pr
         beta(14) = 0.0_pr
         beta(15) = .250_pr
         beta(16) = .20_pr
         beta(17) = -25.0_pr / 108.0_pr
         beta(18) = 0.0_pr
         beta(19) = 0.0_pr
         beta(20) = 125.0_pr / 108.0_pr
         beta(21) = -65.0_pr / 27.0_pr
```

```
beta(22) = 2.0_pr * beta(20)
beta(23) = 31.0_pr / 300.0_pr
beta(24) = 0.0_pr
beta(25) = 0.0_pr
beta(26) = 0.0_pr
beta(27) = 61.0_pr / 225.0_pr
beta(28) = -2.0_pr / 9.0_pr
beta(29) = 13.0_pr / 900.0_pr
beta(30) = 2.0_pr
beta(31) = 0.0_pr
beta(32) = 0.0_pr
beta(33) = -53.0_pr / 6.0_pr
beta(34) = 704.0_pr / 45.0_pr
beta(35) = -107.0_pr / 9.0_pr
beta(36) = 67.0_pr / 90.0_pr
beta(37) = 3.0_pr
beta(38) = -91.0_pr / 108.0_pr
beta(39) = 0.0_pr
beta(40) = 0.0_pr
beta(41) = 23.0_pr / 108.0_pr
beta(42) = -976.0_pr / 135.0_pr
beta(43) = 311.0_pr / 54.0_pr
beta(44) = -19.0_pr / 60.0_pr
beta(45) = 17.0_pr / 6.0_pr
beta(46) = -1.0_pr / 12.0_pr
beta(47) = 2383.0_pr / 4100.0_pr
beta(48) = 0.0_pr
beta(49) = 0.0_pr
beta(50) = -341.0_pr / 164.0_pr
beta(51) = 4496.0_pr / 1025.0_pr
beta(52) = -301.0_pr / 82.0_pr
beta(53) = 2133.0_pr / 4100.0_pr
beta(54) = 45.0_pr / 82.0_pr
beta(55) = 45.0_pr / 164.0_pr
beta(56) = 18.0_pr / 41.0_pr
beta(57) = 3.0_pr / 205.0_pr
beta(58) = 0.0_pr
beta(59) = 0.0_pr
beta(60) = 0.0_pr
beta(61) = 0.0_pr
beta(62) = -6.0_pr / 41.0_pr
beta(63) = -3.0_pr / 205.0_pr
beta(64) = -3.0_pr / 41.0_pr
beta(65) = -beta(64)
beta(66) = -beta(62)
beta(67) = 0.0_pr
beta(68) = -1777.0_pr / 4100.0_pr
beta(69) = 0.0_pr
beta(70) = 0.0_pr
beta(71) = beta(50)
beta(72) = beta(51)
beta(73) = -289.0_pr / 82.0_pr
beta(74) = 2193.0_pr / 4100.0_pr
beta(75) = 51.0_pr / 82.0_pr
beta(76) = 33.0_pr / 164.0_pr
beta(77) = 12.0_pr / 41.0_pr
beta(78) = 0.0_pr
beta(79) = 1.0_pr
```

```
          c(1) = 41.0_pr / 840.0_pr
          c(2) = 0.0_pr
          c(3) = 0.0_pr
          c(4) = 0.0_pr
          c(5) = 0.0_pr
          c(6) = 34.0_pr / 105.0_pr
          c(7) = 9.0_pr / 35.0_pr
          c(8) = c(7)
          c(9) = 9.0_pr / 280.0_pr
          c(10) = c(9)
          c(11) = c(1)

          cp(1) = 0.0_pr
          cp(2) = 0.0_pr
          cp(3) = 0.0_pr
          cp(4) = 0.0_pr
          cp(5) = 0.0_pr
          cp(6) = c(6)
          cp(7) = c(7)
          cp(8) = c(8)
          cp(9) = c(9)
          cp(10) = c(10)
          cp(11) = 0.0_pr
          cp(12) = c(1)
          cp(13) = c(1)

          bFirst = .false.

      endif

9     continue

12    jk = 1

      do 3 j = 1,13

      do 6 l = 1,n
6     b(l) = y(l)
      a = x + alpha(j) * h
      if(j == 1) goto 13
      j1 = j-1
      do 4 k = 1,j1,1
      jk = jk + 1
      bet = beta(jk)* h
      do 4 l = 1,n
4     b(l) = b(l) + bet * r(k,l)
13    continue

      call CalcVectorField(a,b,n,f,acc)

      do 3 l = 1,n
3     r(j,l) = f(l)
      d = 0
      dd = 0
      do 1 l = 1,n
      b(l) = y(l)
      f(l) = y(l)
      do 5 k = 1,11
      bet = h * r(k,l)
```

```
      b(l) = b(l) + bet * c(k)
5     f(l) = f(l) + bet * cp(k)
      f(l) = f(l) + h * (cp(12) * r(12,l) + cp(13) * r(13,l))
      d = d + abs(f(l) - b(l))
1     dd = dd + abs(f(l))
      d = d/n
      fac = ONE + dd * 1.e-2_pr
      e3 = e1 * fac

      if(abs(h) < hmi .OR. d < e3) goto 7

      h = h * 0.9_pr * (e3/d)**0.125_pr
      if(abs(h) < hmi) h = hmi * h/abs(h)
      goto 9

 7    x = x + h

      if(d < e3) d = max(d,e3/256.0_pr)

      h = h * 0.9_pr * (e3/d)**0.125_pr

      if(abs(h) > hmax) h = hmax * h / abs(h)
      if(abs(h) < hmi) h = hmi * h / abs(h)

11    do 10 l = 1,n
10    y(l) = f(l)
      b(1) = d


      return

      end subroutine RungeKutta78

end module NumericalIntegration
```

A.14. Stokes.f90

```
!****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!****************************************************************
! Allocation of arrays for storage of geopotential Stokes
coefficients.
! Reading of Stokes coefficients from input file
!****************************************************************

module Stokes

      use Constants

      implicit none

      real(pr), SAVE, dimension (:,:), allocatable :: clm

      contains

!****************************************************************
subroutine Stokes_AllocDealloc(bAllocate,n_min,n_max,m_min,m_max)
!****************************************************************
! allocate arrays containing gravity field coefficients
!                                                       *
!****************************************************************

      implicit none

      logical, intent(in) :: bAllocate
      integer, intent(in) :: n_min,n_max,m_min,m_max

      integer             :: i1,i2

      if(bAllocate) then

               i1 = min(n_min,max(m_min-1,0))
               i2 = max(n_max,max(m_max-1,0))

      if(.not.allocated(clm)) allocate(clm(i1:i2,i1:i2))
           clm = ZERO

      else
           if(allocated(clm))            deallocate ( clm )
      endif

      return

      end subroutine Stokes_AllocDealloc

!****************************************************************
subroutine
ReadStokesCoeff(iu_MoonModel,MoonModelFile,n_min,n_max,m_min,m_max
)
!****************************************************************
```

```
!     read gravity field model coefficients from file
!*****************************************************************

      implicit none

      integer, intent(in):: iu_MoonModel,n_min,n_max,m_min,m_max
      character*150      :: MoonModelFile

      logical            :: bContinue
      integer            :: l,m,m_1,m1,m2
      real(pr)           :: c,s,sigma_c,sigma_s,coeff
      integer            :: last

      ! read input file:
      open(unit = iu_MoonModel, status = 'old', file =
MoonModelFile, action = 'read')

      bContinue = .true.
      do while(bContinue)

            read(iu_MoonModel, fmt=101, end=10)
l,m,c,s,sigma_c,sigma_s

            if(l.gt.n_max) exit
            if(l.lt.n_min) cycle
            if(m.lt.m_min) cycle
            if(m.gt.m_max) cycle

            m_1 = m-1

            if(m == 0) then
                  ! zonal terms
                  Clm(l,l) = c
            else
                  ! tesseral and sectorial
                  Clm(l,m_1) = c
                  Clm(m_1,l) = s
            endif

            last = l

            if(bContinue) cycle

 10         bContinue = .false.

      end do

      close(iu_MoonModel,status='keep')

101 format(I5,1X,I5,4(1X,E23.16))

    if(last == n_max) return

    write(6,*) '...adding extra Stokes coefficients (Kaula rule of
thumb)'

    do l = last+1,n_max

            coeff = sqrt((TWO * l + ONE)/(l + ONE)) / l / 1.e5_pr
```

```
            m1 = max(m_min - 1,0)
            m2 = m_max - 1

            Clm(l,m1:m2) = coeff
            Clm(m1:m2,l) = coeff

            if(m_min == 0) Clm(l,l) = coeff

      enddo

      return

      end subroutine ReadStokesCoeff
      !
      !
end module Stokes
```

A.15. VectorField.f90

```
!*****************************************************************
! Program name: Orbit_Integration
! Author: Edgar Cardoso
! Tutor: Dr. Elena Fantino
! Date: January 2012
!*****************************************************************

module VectorField

use InputOutput
use Pines
use OtherPerturb

implicit none

contains

!*****************************************************************
    subroutine CalcVectorField(t,x,n,field,acc)
!*****************************************************************
!  Computation of the vector field of the spatial RTBP and, if
required,
!  its variational equations
!
!  The larger primary is at (xmu,0,0) and has mass 1-xmu.
!  The smaller primary is at (xmu-1,0,0) and has mass xmu.
!
!  IN:
!
!    t     RTBP time
!    xmu   RTBP mass parameter
!    x(*)  RTBP variational coordinates (x(1),x(2),...,x(42))
!    n     number of equations (6 or 42 depending on wether the
!           vector field or the full variational flow is
requested)
!
!  OUT:
!
!    field(*)  vector field: the first 6 components store the
!          RTBP equations, the remaining contain the variational
equations
!          by columns.
!
!  NOTE: if n = 6 the variational equations are not computed
!
!*****************************************************************
          implicit none

        ! Arguments
        integer, intent(in)                     :: n
        real(pr), dimension(n), intent(in)      :: x
        real(pr), intent(in)                    :: t
        real(pr), dimension(n), intent(out)     :: field
        real(pr), dimension(3)                  :: acc,
Pines_acc, Earth_acc
```

```
        real(pr), dimension(3)                    :: Sun_acc,
SRP_acc, thrm_acc


        if(n .NE. 6) stop 'CalcVectorField: incorrect vector size'

        field(1) = x(4)
        field(2) = x(5)
        field(3) = x(6)

        call PinesAccel(x, Pines_acc)

        if(Earth3rdbody .OR. Sun3rdbody .OR. SunPressure .OR.
Thermal) then
            call OtherAccel(x, t, Earth_acc, Sun_acc, SRP_acc,
thrm_acc)
        endif

        acc =  Pines_acc + aux1 * Earth_acc + aux2 * Sun_acc +
aux3 * SRP_acc + aux4 * thrm_acc

        field(4) = acc(1)
        field(5) = acc(2)
        field(6) = acc(3)

        if(n .EQ. 6) return

    end subroutine CalcVectorField

end module VectorField
```

ANNEX B.

MATLAB SCRIPTS

B.1 PlotOrbits.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program name: Orbit_Integration
% Author: Edgar Cardoso
% Tutor: Dr. Elena Fantino
% Date: January 2012
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;

inpath = 'C:\Users\Edgar\Desktop\PFC\Programa Pines. E.
FANTINO\Matlab dibujo órbitas\';
outpath = inpath;
str_body1 = 'orbita_n';
nFonts = 20;

str_color1 = '.k';
str_color2 = '.g';
str_color3 = '.r';

set(gca,'FontSize', nFonts);

orbit1 = load(strcat(inpath,'position','_',str_body1,'.txt'));
orbit2 =
load(strcat(inpath,'position','_',str_body1,'_inicial.txt'));
orbit3 =
load(strcat(inpath,'position','_',str_body1,'_final.txt'));

%!!!!
figure(1);

plot_sphere(1.738e6, 0, 0, 0, 1000)

%!!!
hold on;
%!!!
grid on;
%!!!
box on;

%!!!
plot3(orbit1(:,1),orbit1(:,2),orbit1(:,3),str_color1);
```

```
plot3(orbit2(:,1),orbit2(:,2),orbit2(:,3),str_color2);
plot3(orbit3(:,1),orbit3(:,2),orbit3(:,3),str_color3);

    xmax = max(1.738e6,max(orbit1(:,1)));
    xmin = min(-1.738e6,min(orbit1(:,1)));
    ymax = max(1.738e6,max(orbit1(:,2)));
    ymin = min(-1.738e6,min(orbit1(:,2)));
    zmax = max(1.738e6,max(orbit1(:,3)));
    zmin = min(-1.738e6,min(orbit1(:,3)));

xlabel('x [m]','FontSize',nFonts,'Color','k');
ylabel('y [m]','FontSize',nFonts,'Color','k');
zlabel('z [m]','FontSize',nFonts,'Color','k');

axis([xmin xmax ymin ymax zmin zmax 0 10000]);
```