```
#include "C:\My Documents\Programming\tensor++\tensor.hpp"
#include <dos.h>
#define NUM 2
#define STD_SCHW 1
#define PG_SCHW  2
#define ISO_SCHW 3
#define RK2      1
#define ICN      2
#define MAX(a,b) a > b ? a : b
#define MIN(a,b) a < b ? a : b

struct SIM { double min_x; double min_y; double min_z;
             double dx;    double dy;    double dz;
             int    Nx;    int Ny;       int Nz;
             double M;     int metric_choice;
             int num;      int int_choice;
             double dt;    int steps;    int freq;
             double h;     int K;        tensor state;
             int    smooth_derivs;       double H;
             int   kernel_choice;        int RHSOnly;};

void   main(void);
void   get_sim_parms(SIM* sim_parms);
void   get_line(char* line, FILE* file);
void   load_metric(tensor***  a, tensor***  b, tensor***  g,
                   tensor*** da, tensor*** db, tensor*** dg,
                   SIM* sim_parms);
void   std_schw(  tensor a,   tensor b,  tensor g, double* pos, SIM* sim_parms);
void   std_schw_d(tensor da, tensor db, tensor dg, double* pos, SIM* sim_parms);
void   pg_schw(tensor a, tensor b, tensor g, double* pos, SIM* sim_parms);
void   pg_schw_d(tensor da, tensor db, tensor dg, double* pos, SIM* sim_parms);
void   iso_schw(tensor a, tensor b, tensor g, double* pos, SIM* sim_parms);
void   iso_schw_d(tensor da, tensor db, tensor dg, double* pos, SIM* sim_parms);
void   calc_RHS(double* state, double* dstate);
double W(double* r, double* s, SIM* sim_parms);
void   RHS(tensor*** a,   tensor*** b,    tensor*** g,
           tensor*** da,  tensor*** db,   tensor*** dg,
           double* state, double* dstate, SIM* sim_parms);

void main(void)
{
  tensor ***a, ***b, ***g, ***da, ***db, ***dg;
  SIM sim_parms;
  double state[6], dstate[6], half_state[6], t, dt;
  int i, j, k, Nx, Ny, Nz;
  FILE* ephem;
  FILE* log;
  struct dosdate_t sys_d;
  struct dostime_t sys_t;

  //Get the simulation parameters
  get_sim_parms(&sim_parms);
  t = 0;
  dt = sim_parms.dt;

  for( i = 0; i < 6; i++)
    state[i] = sim_parms.state.Val(i);

  Nx = sim_parms.Nx;
  Ny = sim_parms.Ny;
  Nz = sim_parms.Nz;

  //Open output files
  if( sim_parms.RHSOnly == 0 )
  {
    log   = fopen("run_FP_geos.log","w");
```

```
    ephem = fopen("ephem.txt","w");
  }
  else
  {
    log   = fopen("run_FP_RHS.log","w");
    ephem = fopen("rhs.txt","w");
  }

  //Get system date and time and output to the log file
  _dos_getdate(&sys_d);
  _dos_gettime(&sys_t);
  if( sim_parms.RHSOnly == 0 )
  {
    fprintf(log,"**********Subscibe Only Geodesics Run**********\n\n");
  }
  else
  {
    fprintf(log,"**********Subscibe Only RHS Run****************\n\n");
  }
  fprintf(log,"The run started on %d/%d/%d at %2d:%02d:%02d\n",
         sys_d.month, sys_d.day,    sys_d.year,
         sys_t.hour,  sys_t.minute, sys_t.second);


  fprintf(log,"The spatial extent of the grid is %gx%g, %gx%g, %gx%g\n",
          sim_parms.min_x, sim_parms.min_x + (double)(Nx-1)*sim_parms.dx,
          sim_parms.min_y, sim_parms.min_y + (double)(Ny-1)*sim_parms.dy,
          sim_parms.min_z, sim_parms.min_z + (double)(Nz-1)*sim_parms.dz);
  fprintf(log,"The number of grid points is [%d,%d,%d]\n",Nx,Ny,Nz);
  fprintf(log,"The deltas are %16.9lf %16.9lf, %16.9lf\n",
          sim_parms.dx, sim_parms.dy, sim_parms.dz);
  fprintf(log,"The smoothing length is %g\n",sim_parms.h);

  //assume that the indices are ordered x, y, z
  //allocate space for the tensors
  fprintf(log,"Ready to allocate space for the tensors\n");
  a = new tensor** [Nx];
  b = new tensor** [Nx];
  g = new tensor** [Nx];
  if( sim_parms.smooth_derivs == 0 )
  {
    da = new tensor** [Nx];
    db = new tensor** [Nx];
    dg = new tensor** [Nx];
  }
  for( i = 0; i < Nx; i++ )
  {
    a[i] = new tensor* [Ny];
    b[i] = new tensor* [Ny];
    g[i] = new tensor* [Ny];
    if( sim_parms.smooth_derivs == 0 )
    {
      da[i] = new tensor* [Ny];
      db[i] = new tensor* [Ny];
      dg[i] = new tensor* [Ny];
    }
    for( j = 0; j < Ny; j++ )
    {
      a[i][j] = new tensor [Nz];
      b[i][j] = new tensor [Nz];
      g[i][j] = new tensor [Nz];
      if( sim_parms.smooth_derivs == 0 )
       {
         da[i][j] = new tensor [Nz];
         db[i][j] = new tensor [Nz];
         dg[i][j] = new tensor [Nz];
```

```
                }
              for( k = 0; k < Nz; k++ )
              {
                a[i][j][k].Resize(1,1);
                b[i][j][k].Resize(1,3);
                g[i][j][k].Resize(2,3,3);
                if( sim_parms.smooth_derivs == 0 )
                 {
                   da[i][j][k].Resize(1,3);
                   db[i][j][k].Resize(2,3,3);
                   dg[i][j][k].Resize(3,3,3,3);
                 }
              }
            }
          }
          _dos_gettime(&sys_t);
          fprintf(log,"Grid completely allocated at %2d:%02d:%02d - preparing to load\n",
                  sys_t.hour, sys_t.minute, sys_t.second);

          //now load the values for the lapse, shift, and 3-metric
          load_metric(a,b,g,da,db,dg,&sim_parms);

          _dos_gettime(&sys_t);
          fprintf(log,"Grid loaded at %2d:%02d:%02d\n",
                  sys_t.hour, sys_t.minute, sys_t.second);


          if( sim_parms.RHSOnly == 0 )
          {
            for(i = 0; i <= sim_parms.steps; i++)
            {
              printf(".");
              //1) output the ephem
              if( i%sim_parms.freq == 0 )
              {
                fprintf(ephem,"%16.9lf %16.9lf %16.9lf %16.9lf %16.9lf %16.9lf %16.9lf %16.9lf
        \n",
                        t,state[0],state[1],state[2],state[3],state[4],state[5],sim_parms.H);
                printf("*");
              }

              //2) take a half step
              RHS(a, b, g, da, db, dg, state, dstate, &sim_parms);
              for( j = 0; j < 6; j++) half_state[j] = state[j] + 0.5 * dt * dstate[j];

              //3) take a full step
              RHS(a, b, g, da, db, dg, half_state, dstate, &sim_parms);
              for( j = 0; j < 6; j++) state[j] = state[j] + dt * dstate[j];

              //4) increment the time
              t = t + dt;
            }

            //5) output the final state ephem
            fprintf(ephem,"%16.9lf %16.9lf %16.9lf %16.9lf %16.9lf %16.9lf %16.9lf %16.9lf\n",
                    t,state[0],state[1],state[2],state[3],state[4],state[5],sim_parms.H);
          }
          else
          {
            RHS(a, b, g, da, db, dg, state, dstate, &sim_parms);
            fprintf(ephem,"%1.16g\n%1.16g\n%1.16g\n%1.16g\n%1.16g\n%1.16g",
                    dstate[0],dstate[1],dstate[2],dstate[3],
                    dstate[4],dstate[5],dstate[6]);
          }
          _dos_gettime(&sys_t);
          fprintf(log,"Run finished at %2d:%02d:%02d\n",
```

```
              sys_t.hour, sys_t.minute, sys_t.second);

}
//*********************************************************************
//load_metric
//*********************************************************************
void   load_metric(tensor***  a, tensor***  b, tensor***  g,
                   tensor*** da, tensor*** db, tensor*** dg,
                   SIM* sim_parms)
{
   int i, j, k, Nx, Ny, Nz;
   double pos[3];

   Nx = sim_parms->Nx;
   Ny = sim_parms->Ny;
   Nz = sim_parms->Nz;


   for( i = 0; i < Nx; i++)
   {
     for( j = 0; j < Ny; j++)
       for( k = 0; k < Nz; k++)
         {
           pos[0] = sim_parms->min_x + (double)i * sim_parms->dx;
           pos[1] = sim_parms->min_y + (double)j * sim_parms->dy;
           pos[2] = sim_parms->min_z + (double)k * sim_parms->dz;
           switch(sim_parms->metric_choice)
           {
             case STD_SCHW:
               std_schw(a[i][j][k],b[i][j][k],g[i][j][k],pos,sim_parms);
               if( sim_parms->smooth_derivs == 0)
                 std_schw_d(da[i][j][k],db[i][j][k],dg[i][j][k],pos,sim_parms);
               break;
             case PG_SCHW:
               pg_schw(a[i][j][k],b[i][j][k],g[i][j][k],pos,sim_parms);
               if( sim_parms->smooth_derivs == 0)
                 pg_schw_d(da[i][j][k],db[i][j][k],dg[i][j][k],pos,sim_parms);
               break;
             case ISO_SCHW:
               iso_schw(a[i][j][k],b[i][j][k],g[i][j][k],pos,sim_parms);
               if( sim_parms->smooth_derivs == 0)
                 iso_schw_d(da[i][j][k],db[i][j][k],dg[i][j][k],pos,sim_parms);
               break;
           }
         }
   }
}

//*********************************************************************
//std_schw
//*********************************************************************
void std_schw(tensor a, tensor b, tensor g, double* pos, SIM* sim_parms)
{
   double M, r, M2_r, Q;

   M = sim_parms->M;
   r = sqrt( pos[0] * pos[0] + pos[1] * pos[1] + pos[2] * pos[2] );
   if( r > 2.0 *M )
   {
     M2_r = 2.0*M/r;
     Q = M2_r/( r * r * ( 1.0 - M2_r ) );

     a.Set(sqrt(fabs(1.0 - M2_r)),0);

     g.Set(1.0 + pos[0]*pos[0]*Q,0,0);
     g.Set(      pos[0]*pos[1]*Q,0,1);
```

```
      g.Set(      pos[0]*pos[2]*Q,0,2);
      g.Set(      pos[1]*pos[0]*Q,1,0);
      g.Set(1.0 + pos[1]*pos[1]*Q,1,1);
      g.Set(      pos[1]*pos[2]*Q,1,2);
      g.Set(      pos[2]*pos[0]*Q,2,0);
      g.Set(      pos[2]*pos[1]*Q,2,1);
      g.Set(1.0 + pos[2]*pos[2]*Q,2,2);
    }
}

//**********************************************************************
//std_schw_d
//**********************************************************************
void std_schw_d(tensor da, tensor db, tensor dg, double* pos, SIM* sim_parms)
{
   double M2, r, x, y, z, inv_r, inv_r3, a, inv_a, Q, P;

   M2 = 2.0*sim_parms->M;
   r = sqrt( pos[0] * pos[0] + pos[1] * pos[1] + pos[2] * pos[2] );

   if( r > M2 )
   {
     inv_r = 1.0/r;
     inv_r3 = inv_r * inv_r * inv_r;
     Q = M2*inv_r3/(1.0 - M2*inv_r);
     P = (3.0*inv_r*inv_r + Q);
     x = pos[0];
     y = pos[1];
     z = pos[2];


     dg.Set(2.0*x - P*x*x*x,  0,0,0);
     dg.Set(-P*x*x*y       ,  0,0,1);
     dg.Set(-P*x*x*z       ,  0,0,2);
     dg.Set(y - P*x*x*y    ,  0,1,0);
     dg.Set(x - P*x*y*y    ,  0,1,1);
     dg.Set(-P*x*y*z       ,  0,1,2);
     dg.Set(z - P*x*x*z    ,  0,2,0);
     dg.Set(-P*x*y*z       ,  0,2,1);
     dg.Set(x - P*x*z*z    ,  0,2,2);

     dg.Set(y - P*x*x*y    ,  1,0,0);
     dg.Set(x - P*x*y*y    ,  1,0,1);
     dg.Set(-P*x*y*z       ,  1,0,2);
     dg.Set(-P*x*y*y       ,  1,1,0);
     dg.Set(2.0*y - P*y*y*y,  1,1,1);
     dg.Set(-P*y*y*z       ,  1,1,2);
     dg.Set(-P*x*y*z       ,  1,2,0);
     dg.Set(z - P*y*y*z    ,  1,2,1);
     dg.Set(y - P*y*z*z    ,  1,2,2);

     dg.Set(z - P*x*x*z    ,  2,0,0);
     dg.Set(-P*x*y*z       ,  2,0,1);
     dg.Set(x - P*x*z*z    ,  2,0,2);
     dg.Set(-P*x*x*z       ,  2,1,0);
     dg.Set(z - P*y*y*z    ,  2,1,1);
     dg.Set(y - P*y*z*z    ,  2,1,2);
     dg.Set(-P*x*z*z       ,  2,2,0);
     dg.Set(-P*y*z*z       ,  2,2,1);
     dg.Set(2.0*z - P*z*z*z,  2,2,2);
     dg <= Q * dg;

     a     = sqrt(1.0 - M2*inv_r);
     inv_a = 1.0/a;
     da.Set(0.5*M2*inv_r3*inv_a*x,0);
     da.Set(0.5*M2*inv_r3*inv_a*y,1);
```

```
      da.Set(0.5*M2*inv_r3*inv_a*z,2);
    }
  }

  //*********************************************************************
  //pg_schw
  //*********************************************************************
  void pg_schw(tensor a, tensor b, tensor g, double* pos, SIM* sim_parms)
  {
    double M, r, M2_r3, Q;

    M = sim_parms->M;
    r = sqrt( pos[0] * pos[0] + pos[1] * pos[1] + pos[2] * pos[2] );
    if( r > 1e-10 )
    {
      M2_r3 = 2.0*M/(r*r*r);
      Q = sqrt( M2_r3 );

      a.Set(1.0,0);
      b.Set(-pos[0]*Q,0);
      b.Set(-pos[1]*Q,1);
      b.Set(-pos[2]*Q,2);

      g.Set(1.0,0,0);
      g.Set(0.0,0,1);
      g.Set(0.0,0,2);
      g.Set(0.0,1,0);
      g.Set(1.0,1,1);
      g.Set(0.0,1,2);
      g.Set(0.0,2,0);
      g.Set(0.0,2,1);
      g.Set(1.0,2,2);
    }
  }

  //*********************************************************************
  //pg_schw_d
  //*********************************************************************
  void pg_schw_d(tensor da, tensor db, tensor dg, double* pos, SIM* sim_parms)
  {
    double M2, r, inv_r, inv_r2, inv_r3, x, y, z, Q;

    M2 = 2.0*sim_parms->M;
    r = sqrt( pos[0] * pos[0] + pos[1] * pos[1] + pos[2] * pos[2] );
    if( r > 1e-10 )
    {
      inv_r  = 1.0/r;
      inv_r2 = inv_r*inv_r;
      inv_r3 = inv_r2*inv_r;
      x = pos[0];
      y = pos[1];
      z = pos[2];
      Q = sqrt( M2 * inv_r3 );

      db.Set(1.5*x*x*inv_r2 - 1.0 ,0,0);
      db.Set(1.5*x*y*inv_r2       ,0,1);
      db.Set(1.5*x*z*inv_r2       ,0,2);
      db.Set(1.5*y*x*inv_r2       ,1,0);
      db.Set(1.5*y*y*inv_r2 - 1.0 ,1,1);
      db.Set(1.5*y*z*inv_r2       ,1,2);
      db.Set(1.5*z*x*inv_r2       ,2,0);
      db.Set(1.5*z*y*inv_r2       ,2,1);
      db.Set(1.5*z*z*inv_r2 - 1.0 ,2,2);
      db <= Q*db;
    }
  }
```

```
//*******************************************************************
//iso_schw
//*******************************************************************
void iso_schw(tensor a, tensor b, tensor g, double* pos, SIM* sim_parms)
{
   double M, r, M_2r, op_M_2r, om_M_2r, Q, R;

   M = sim_parms->M;
   r = sqrt( pos[0] * pos[0] + pos[1] * pos[1] + pos[2] * pos[2] );
   if( r > 1e-10 )
   {
     M_2r    = 0.5*M/r;
     op_M_2r = 1.0 + M_2r;
     om_M_2r = 1.0 - M_2r;
     Q       = om_M_2r / op_M_2r;
     R       = op_M_2r * op_M_2r * op_M_2r * op_M_2r;

     a.Set(Q,0);

     g.Set(R  ,0,0);
     g.Set(0.0,0,1);
     g.Set(0.0,0,2);
     g.Set(0.0,1,0);
     g.Set(R  ,1,1);
     g.Set(0.0,1,2);
     g.Set(0.0,2,0);
     g.Set(0.0,2,1);
     g.Set(R  ,2,2);
   }
}

//*******************************************************************
//iso_schw_d
//*******************************************************************
void iso_schw_d(tensor da, tensor db, tensor dg, double* pos, SIM* sim_parms)
{
   double M, M_2r, r, x, y, z, inv_r, inv_r3, op_M_2r, inv_op_2, M_inv_r3, op_3;

   M = sim_parms->M;
   r = sqrt( pos[0] * pos[0] + pos[1] * pos[1] + pos[2] * pos[2] );

   if( r > 1e-10 )
   {
     M_2r     = 0.5*M/r;
     inv_r    = 1.0/r;
     inv_r3   = inv_r * inv_r * inv_r;
     op_M_2r  = 1.0 + M_2r;
     op_3     = op_M_2r * op_M_2r * op_M_2r;
     inv_op_2 = 1.0/op_M_2r/op_M_2r;
     M_inv_r3 = M*inv_r3;
     x = pos[0];
     y = pos[1];
     z = pos[2];

     da.Set(x*M_inv_r3*inv_op_2,0);
     da.Set(y*M_inv_r3*inv_op_2,1);
     da.Set(z*M_inv_r3*inv_op_2,2);

     dg.Set(-2.0*op_3*M_inv_r3*x,  0,0,0);
     dg.Set(-2.0*op_3*M_inv_r3*y,  0,0,1);
     dg.Set(-2.0*op_3*M_inv_r3*z,  0,0,2);
     dg.Set(0.0                 ,  0,1,0);
     dg.Set(0.0                 ,  0,1,1);
     dg.Set(0.0                 ,  0,1,2);
     dg.Set(0.0                 ,  0,2,0);
```

```
      dg.Set(0.0                 ,  0,2,1);
      dg.Set(0.0                 ,  0,2,2);

      dg.Set(0.0                 ,  1,0,0);
      dg.Set(0.0                 ,  1,0,1);
      dg.Set(0.0                 ,  1,0,2);
      dg.Set(-2.0*op_3*M_inv_r3*x,  1,1,0);
      dg.Set(-2.0*op_3*M_inv_r3*y,  1,1,1);
      dg.Set(-2.0*op_3*M_inv_r3*z,  1,1,2);
      dg.Set(0.0                 ,  1,2,0);
      dg.Set(0.0                 ,  1,2,1);
      dg.Set(0.0                 ,  1,2,2);

      dg.Set(0.0                 ,  2,0,0);
      dg.Set(0.0                 ,  2,0,1);
      dg.Set(0.0                 ,  2,0,2);
      dg.Set(0.0                 ,  2,1,0);
      dg.Set(0.0                 ,  2,1,1);
      dg.Set(0.0                 ,  2,1,2);
      dg.Set(-2.0*op_3*M_inv_r3*x,  2,2,0);
      dg.Set(-2.0*op_3*M_inv_r3*y,  2,2,1);
      dg.Set(-2.0*op_3*M_inv_r3*z,  2,2,2);
    }
}

//***********************************************************************
//smoother
//***********************************************************************
double W(double* r, double* s, SIM* sim_parms)
{
   double y2, val, ih3;

   y2 = (  (r[0] - s[0])*(r[0] - s[0])
         + (r[1] - s[1])*(r[1] - s[1])
         + (r[2] - s[2])*(r[2] - s[2]) ) / ( sim_parms->h * sim_parms->h );

   ih3 = 1.0 / ( sim_parms->h * sim_parms->h * sim_parms->h );

   if ( y2 > 1.0 )
   {
     val = 0.0;
   }
   else
   {
     switch (sim_parms->kernel_choice)
     {
       case 2 :
         val = 1.0444543140405631410 * ih3 * ( 1.0 - y2 ) * ( 1.0 - y2);
         break;
       case 3 :
         val = 1.5666814710608447114 * ih3 *
               ( 1.0 - y2 )*( 1.0 - y2 )*( 1.0 - y2 );
         break;
       case 4 :
         val = 2.1541870227086614782 * ih3 *
               ( 1.0 - y2 )*( 1.0 - y2 )*( 1.0 - y2 )*( 1.0 - y2 );
         break;
       case 999 :
         val = exp(-9.0*y2);
         break;
       default:
         val = 1.5666814710608447114 * ih3 *
               ( 1.0 - y2 )*( 1.0 - y2 )*( 1.0 - y2 );
         break;
     }
   }
```

```
   return val;
}

//*********************************************************************
//smoother derivative
//*********************************************************************
void DW(double* r, double* s, tensor dW, SIM* sim_parms)
{
   double ih2, ih5, y2, N;

   ih2 = 1.0/(sim_parms->h * sim_parms->h);
   ih5 = ih2 * ih2 / sim_parms->h;

   y2 = (  (r[0] - s[0])*(r[0] - s[0])
         + (r[1] - s[1])*(r[1] - s[1])
         + (r[2] - s[2])*(r[2] - s[2]) ) * ih2;

   if( y2 > 1.0 )
   {
      dW <= 0.0 * dW;
   }
   else
   {
      switch ( sim_parms->kernel_choice)
      {
        case 2 :
          N = -4.1778172561622525638 * ih5 * ( 1.0 - y2 );
          break;
        case 3 :
          N = -9.4000888263650682686 * ih5 * ( 1.0 - y2 ) * ( 1.0 - y2 );
          break;
        case 4 :
          N = -17.233496181669291826 * ih5 *
            ( 1.0 - y2 ) * ( 1.0 - y2 ) * ( 1.0 - y2 );
          break;
        case 999 :
          N = -18.0 * ih2 * exp(-9.0*y2);
          break;
        default:
          N = -9.4000888263650682686 * ih5 * ( 1.0 - y2 ) * ( 1.0 - y2 );
          break;
      }
      dW.Set(N * ( r[0] - s[0] ),0);
      dW.Set(N * ( r[1] - s[1] ),1);
      dW.Set(N * ( r[2] - s[2] ),2);
   }
}

//*********************************************************************
//smooth metric
//*********************************************************************
void RHS(tensor*** a,   tensor*** b,    tensor*** g,
         tensor*** da,  tensor*** db,   tensor*** dg,
         double* state, double* dstate, SIM* sim_parms)
{
  double h, pos[3], w, Lambda, detG, inv_L, H, sumW, sumW_inv;
  int i, j, k, m, n, o, min_ix, min_iy, min_iz, max_ix, max_iy, max_iz;
  int Numx, Numy, Numz;

  tensor A(1,1),  B(1,3),       G(2,3,3),     Ginv(2,3,3),     Bup(1,3);
  tensor dA(1,3), dB(2,3,3),    dG(3,3,3,3), dGinv(3,3,3,3), dBup(2,3,3);
  tensor dW(1,3), dW_sum(1,3), r(1,3),       u(1,3),          dr(1,3),    du(1,3);

//  FILE* mojo;
//  char *dump;
```

```
//  mojo = fopen("debug.txt","w");
//  dump = NULL;

    for(i = 0; i < 2; i++)
    {
      r.Set(state[i],i);
      u.Set(state[i+3],i);
    }

    h = sim_parms->h;

    //find the closest, lowest corner
    m   = floor( ( state[0] - sim_parms->min_x ) / sim_parms->dx );
    n   = floor( ( state[1] - sim_parms->min_y ) / sim_parms->dy );
    o   = floor( ( state[2] - sim_parms->min_z ) / sim_parms->dz );
    Numx = ceil( h / sim_parms->dx );
    Numy = ceil( h / sim_parms->dy );
    Numz = ceil( h / sim_parms->dz );
    min_ix = MAX(m-Numx,0);
    min_iy = MAX(n-Numy,0);
    min_iz = MAX(o-Numz,0);
    max_ix = MIN(m+Numx,sim_parms->Nx-1);
    max_iy = MIN(n+Numy,sim_parms->Ny-1);
    max_iz = MIN(o+Numz,sim_parms->Nz-1);


    sumW = 0.0;

//  fprintf(mojo,"m = %d, n = %d, o = %d, Num = %d\n",m,n,o,Num);

    for(i = min_ix; i < max_ix; i++)
    {
      pos[0] = sim_parms->min_x + i * sim_parms->dx;
      for(j = min_iy; j < max_iy; j++)
      {
        pos[1] = sim_parms->min_y + j * sim_parms->dy;
        for(k = min_iz; k < max_iz; k++)
        {
          pos[2] = sim_parms->min_z + k * sim_parms->dz;
          w = W(state,pos,sim_parms);
          DW(state, pos, dW, sim_parms);
          sumW = sumW + w;
          dW_sum <= dW_sum + dW;

          A <= A + w*a[i][j][k];
          B <= B + w*b[i][j][k];
          G <= G + w*g[i][j][k];

          if ( sim_parms->smooth_derivs == 0 )
          {
            dA <= dA + w*da[i][j][k];
            dB <= dB + w*db[i][j][k];
            dG <= dG + w*dg[i][j][k];
          }

          if ( sim_parms->smooth_derivs == 1 )
          {
            dA <= dA + a[i][j][k]*dW;
            dB <= dB + b[i][j][k]*dW;
            dG <= dG + g[i][j][k]*dW;
          }

//          fprintf(mojo,"i = %d, j = %d, k = %d sumW = %g\n",i,j,k,sumW);
//            fprintf(mojo,"pos = %8.6f, %8.6f, %8.6f \n",pos[0],pos[1],pos[2]);
//            fprintf(mojo,"w = %8.6f\n",w);
```

```
        //          dW.print("dW",&dump);
        //          fprintf(mojo,"dW\n");
        //          fprintf(mojo,"%s",dump);
        //          dW_sum.print("dW_sum",&dump);
        //          fprintf(mojo,"dW_sum\n");
        //          fprintf(mojo,"%s",dump);

                  //a[i][j][k].print("a[i][j][k]",&dump);
                  //fprintf(mojo,"a[i][j][k]\n");
                  //fprintf(mojo,"%s",dump);

                  //g[i][j][k].print("g[i][j][k]",&dump);
                  //fprintf(mojo,"g[i][j][k]\n");
                  //fprintf(mojo,"%s",dump);

                  //A.print("A",&dump);
                  //fprintf(mojo,"A\n");
                  //fprintf(mojo,"%s",dump);

                  //G.print("G",&dump);
                  //fprintf(mojo,"G\n");
                  //fprintf(mojo,"%s",dump);

                  //dA.print("dA",&dump);
                  //fprintf(mojo,"dA\n");
                  //fprintf(mojo,"%s",dump);

                  //dG.print("dG",&dump);
                  //fprintf(mojo,"dG\n");
                  //fprintf(mojo,"%s",dump);
              }
            }
          }

                sumW_inv = 1.0/sumW;
                A <= sumW_inv * A;
                B <= sumW_inv * B;
                G <= sumW_inv * G;

                if ( sim_parms->smooth_derivs == 0 )
                {
                  dA <= sumW_inv*dA;
                  dB <= sumW_inv*dB;
                  dG <= sumW_inv*dG;
                }
                if ( sim_parms->smooth_derivs == 1 )
                {
                  dA <= sumW_inv*(dA - A*dW_sum);
                  dB <= sumW_inv*(dB - B*dW_sum);
                  dG <= sumW_inv*(dG - G*dW_sum);
                }


          //Form the inverse of the smoothed metric
          detG =   G.Val(0,0)*(G.Val(1,1)*G.Val(2,2) - G.Val(1,2)*G.Val(2,1))
                 + G.Val(0,1)*(G.Val(2,0)*G.Val(1,2) - G.Val(1,0)*G.Val(2,2))
                 + G.Val(0,2)*(G.Val(1,0)*G.Val(2,1) - G.Val(1,1)*G.Val(2,0));

          Ginv.Set( G.Val(1,1)*G.Val(2,2) - G.Val(1,2)*G.Val(2,1),0,0);
          Ginv.Set( G.Val(0,2)*G.Val(2,1) - G.Val(0,1)*G.Val(2,2),0,1);
          Ginv.Set( G.Val(0,1)*G.Val(1,2) - G.Val(0,2)*G.Val(1,1),0,2);
          Ginv.Set( G.Val(1,2)*G.Val(2,0) - G.Val(1,0)*G.Val(2,2),1,0);
          Ginv.Set( G.Val(0,0)*G.Val(2,2) - G.Val(0,2)*G.Val(2,0),1,1);
          Ginv.Set( G.Val(0,2)*G.Val(1,0) - G.Val(0,0)*G.Val(1,2),1,2);
          Ginv.Set( G.Val(1,0)*G.Val(2,1) - G.Val(1,1)*G.Val(2,0),2,0);
          Ginv.Set( G.Val(0,1)*G.Val(2,0) - G.Val(0,0)*G.Val(2,1),2,1);
```

```
          Ginv.Set( G.Val(0,0)*G.Val(1,1) - G.Val(0,1)*G.Val(1,0),2,2);

          Ginv <= 1.0/detG * Ginv;

          //now raise the index on the shift
          Bup <= Ginv.Contract(B,1,0);

          //now calculate the tensor needed to the equations of motion
          dGinv <= -1.0 * Ginv.Contract(dG.Contract(Ginv,1,0),1,0);

          dBup  <= dGinv.Contract(B,1,0) + Ginv.Contract(dB,1,0);

          //now calculate the RHS
          Lambda = sqrt( 1.0 + u.Contract(Ginv.Contract(u,1,0),0,0).Val(0) );
          inv_L = 1.0/Lambda;

          dr <= -1.0*Bup + A.Val(0)*inv_L*Ginv.Contract(u,1,0);

          du <= u.Contract(dBup,0,0) - Lambda * dA
                   - A.Val(0) * 0.5 * inv_L * u.Contract(dGinv.Contract(u,2,0),0,0);

          //Calculate the conserved quantities
          H = -Bup.Contract(u,0,0).Val(0) + A.Val(0)*Lambda;

          //pack the state
          for( i = 0; i < 3; i++)
          {
            dstate[i]   = dr.Val(i);
            dstate[i+3] = du.Val(i);
          }
          sim_parms->H = H;

        //        fprintf(mojo,"sumW = %8.6f\n",sumW);
        //
        //        A.print("A",&dump);
        //        fprintf(mojo,"A\n");
        //        fprintf(mojo,"%s",dump);
        //
        //        B.print("B",&dump);
        //        fprintf(mojo,"B\n");
        //        fprintf(mojo,"%s",dump);
        //
        //        G.print("G",&dump);
        //        fprintf(mojo,"G\n");
        //        fprintf(mojo,"%s",dump);
        //
        //        dA.print("dA",&dump);
        //        fprintf(mojo,"dA\n");
        //        fprintf(mojo,"%s",dump);
        //
        //        dB.print("dB",&dump);
        //        fprintf(mojo,"dB\n");
        //        fprintf(mojo,"%s",dump);
        //
        //        dG.print("dG",&dump);
        //        fprintf(mojo,"dG\n");
        //        fprintf(mojo,"%s",dump);
        //
        //  fprintf(mojo,"detG = %8.6f\n",detG);
        //
        //  Ginv.print("Ginv",&dump);
        //  fprintf(mojo,"Ginv\n");
        //  fprintf(mojo,"%s",dump);
        //
        //  Bup.print("Bup",&dump);
        //  fprintf(mojo,"Bup\n");
```

```
//  fprintf(mojo,"%s",dump);
//
//  dGinv.print("dGinv",&dump);
//  fprintf(mojo,"dGinv\n");
//  fprintf(mojo,"%s",dump);
//
//  dBup.print("dBup",&dump);
//  fprintf(mojo,"dBup\n");
//  fprintf(mojo,"%s",dump);

//  fprintf(mojo,"H = %8.6f\n",H);
//  fprintf(mojo,"%s",dump);
//  fprintf(mojo,"Lambda = %8.6f\n",Lambda);
//
//  dr.print("dr",&dump);
//  fprintf(mojo,"dr\n");
//  fprintf(mojo,"%s",dump);
//
//  du.print("du",&dump);
//  fprintf(mojo,"du\n");
//  fprintf(mojo,"%s",dump);
}

//*********************************************************************
//get_sim_parms
//*********************************************************************
void   get_sim_parms(SIM* sim_parms)
{
   FILE* input_file;
   char  line[80];

   input_file = fopen("subscribe_geodesics_parms.txt","r");

   sim_parms->state.Resize(1,6);

   //get min_x, min_y, min_z
   get_line(line, input_file);
   get_line(line, input_file);
   sim_parms->min_x = atof(line);
   get_line(line, input_file);
   sim_parms->min_y = atof(line);
   get_line(line, input_file);
   sim_parms->min_z = atof(line);

   //get grid spacing dx, dy, dz
   get_line(line, input_file);
   get_line(line, input_file);
   sim_parms->dx = atof(line);
   get_line(line, input_file);
   sim_parms->dy = atof(line);
   get_line(line, input_file);
   sim_parms->dz = atof(line);

   //get the lattice size
   get_line(line, input_file);
   get_line(line, input_file);
   sim_parms->Nx = atoi(line);
   get_line(line, input_file);
   sim_parms->Ny = atoi(line);
   get_line(line, input_file);
   sim_parms->Nz = atoi(line);

   //get the mass
   get_line(line, input_file);
   get_line(line, input_file);
   sim_parms->M = atof(line);
```

```
   //get the metric choice
   get_line(line, input_file);
   get_line(line, input_file);
   sim_parms->metric_choice = atoi(line);

   //get integration parameters
   get_line(line, input_file);
   get_line(line, input_file);
   sim_parms->dt = atof(line);
   get_line(line, input_file);
   sim_parms->steps = atoi(line);
   get_line(line, input_file);
   sim_parms->freq = atoi(line);

   //get integration method
   get_line(line, input_file);
   get_line(line, input_file);
   sim_parms->int_choice = atoi(line);

   //get smoothing parameters
   get_line(line, input_file);
   get_line(line, input_file);
   sim_parms->h = atof(line);
   sim_parms->K = ceil(sim_parms->h/sim_parms->dx);

   //get the initial state of the particle
   get_line(line, input_file);
   get_line(line, input_file);
   sim_parms->state.Set(atof(line),0);
   get_line(line, input_file);
   sim_parms->state.Set(atof(line),1);
   get_line(line, input_file);
   sim_parms->state.Set(atof(line),2);
   get_line(line, input_file);
   sim_parms->state.Set(atof(line),3);
   get_line(line, input_file);
   sim_parms->state.Set(atof(line),4);
   get_line(line, input_file);
   sim_parms->state.Set(atof(line),5);

   //get the flag for smoothing of derivatives
   get_line(line, input_file);
   get_line(line, input_file);
   sim_parms->smooth_derivs = atoi(line);

   //get the flag for kernel choice
   get_line(line, input_file);
   get_line(line, input_file);
   sim_parms->kernel_choice = atoi(line);

   //get the flag for RHS choice
   get_line(line, input_file);
   get_line(line, input_file);
   sim_parms->RHSOnly = atoi(line);


}

//**********************************************************************
//get_line
//**********************************************************************
void get_line(char* line, FILE* file)
{
   char c;
   int counter =0, flag = 0;
```

```
    while( flag != 1)
    {
      fread(&c, 1, 1, file);
      if( c == '\n' )
      {
        flag = 1;
      }
      else
      {
        line[counter] = c;
        counter++;
      }
    }
    line[counter] = '\n';
 }
```