```
#include "rk4_solver.hh"
#include <iostream>
#include <iomanip>
#include <math.h>
#include <ctime>
#include <cstdlib>
#include <cstdio>
#include <fstream>
#include <string>
#include <omp.h>

void rk4::compute_xx(double t_, double* x_, double* y_, double* theta_, double* outp
utX, double* outputY, double* output_theta){
  #pragma omp parallel for
  for (int i=0; i<N; i++){
    outputX[i] = 0.;
    outputY[i] = 0.;
    output_theta[i] = 0.;
    double sumx = 0.;
    double sumy = 0.;
    double sumtheta = 0.;
    for (int j=0; j<N; j++){
      if (j!=i){
        double norm2 = (x_[j]-x_[i])*(x_[j]-x_[i]) + (y_[j]-y_[i])*(y_[j]-y_[i]);
        double norm = sqrt(norm2);
        sumx += ((x_[j]-x_[i])/norm)*(AA+J*cos(theta_[j]-theta_[i])) - BB*((x_[j]-x_
[i])/norm2);
        sumy += ((y_[j]-y_[i])/norm)*(AA+J*cos(theta_[j]-theta_[i])) - BB*((y_[j]-y_
[i])/norm2);
        sumtheta += sin(theta_[j]-theta_[i])/norm;
      }
    }
    outputX[i] += (1./float(N))*sumx;
    outputY[i] += (1./float(N))*sumy;
    output_theta[i] += (float(K)/float(N))*sumtheta;
  }
}


void rk4::compute_Gs(double t, double* Gs_x, double* ff_x, double* Gs_y, double* ff_
y, double* Gs_theta, double* ff_theta){
  // First, we calculate G1:
#pragma omp parallel for
  for(int i=0; i<(N); i++){
    Gs_x[i]=x0[i];
    Gs_x[i+1*N]=x0[i];
    Gs_x[i+2*N]=x0[i];
    Gs_x[i+3*N]=x0[i];
    Gs_x[i+4*N]=x0[i];
    Gs_y[i]=y0[i];
    Gs_y[i+1*N]=y0[i];
    Gs_y[i+2*N]=y0[i];
    Gs_y[i+3*N]=y0[i];
    Gs_y[i+4*N]=y0[i];
    Gs_theta[i]=theta0[i];
    Gs_theta[i+1*N]=theta0[i];
    Gs_theta[i+2*N]=theta0[i];
    Gs_theta[i+3*N]=theta0[i];
    Gs_theta[i+4*N]=theta0[i];
  }
  // Then, we compute f(G1):
  compute_xx(t+C[0]*h_step, x0, y0, theta0, ff_x, ff_y, ff_theta);

  // Calculating G2:
#pragma omp parallel for
  for(int i=0; i<N; i++){
    Gs_x[i+1*N]     += A[0]*h_step*ff_x[i];
    Gs_y[i+1*N]     += A[0]*h_step*ff_y[i];
    Gs_theta[i+1*N] += A[0]*h_step*ff_theta[i];
  }

  // Computing f(G2):
  compute_xx(t+C[1]*h_step, (Gs_x+1*N), (Gs_y+1*N), (Gs_theta+1*N), (ff_x+1*N), (ff_
```

```
y+1*N), (ff_theta+1*N));

  // Calculating G3:
  for (int i=0; i<N; i++){
    Gs_x[i+2*N]     += A[1]*h_step*ff_x[i];
    Gs_x[i+2*N]     += A[2]*h_step*ff_x[i+1*N];
    Gs_y[i+2*N]     += A[1]*h_step*ff_y[i];
    Gs_y[i+2*N]     += A[2]*h_step*ff_y[i+1*N];
    Gs_theta[i+2*N] += A[1]*h_step*ff_theta[i];
    Gs_theta[i+2*N] += A[2]*h_step*ff_theta[i+1*N];
  }

  // Computing f(G3):
  compute_xx(t+C[2]*h_step, (Gs_x+2*N), (Gs_y+2*N), (Gs_theta+2*N), (ff_x+2*N), (ff_
y+2*N), (ff_theta+2*N));


  // Calculating G4:
#pragma omp parallel for
  for (int i=0; i<N; i++){
    Gs_x[i+3*N]     += A[3]*h_step*ff_x[i];
    Gs_x[i+3*N]     += A[4]*h_step*ff_x[i+1*N];
    Gs_x[i+3*N]     += A[5]*h_step*ff_x[i+2*N];
    Gs_y[i+3*N]     += A[3]*h_step*ff_y[i];
    Gs_y[i+3*N]     += A[4]*h_step*ff_y[i+1*N];
    Gs_y[i+3*N]     += A[5]*h_step*ff_y[i+2*N];
    Gs_theta[i+3*N] += A[3]*h_step*ff_theta[i];
    Gs_theta[i+3*N] += A[4]*h_step*ff_theta[i+1*N];
    Gs_theta[i+3*N] += A[5]*h_step*ff_theta[i+2*N];
  }

  // Computing f(G4):
  compute_xx(t+C[3]*h_step, (Gs_x+3*N), (Gs_y+3*N), (Gs_theta+3*N), (ff_x+3*N), (ff_
y+3*N), (ff_theta+3*N));


  // Calculating G5:
#pragma omp parallel for
  for (int i=0; i<N; i++){
    Gs_x[i+4*N]     += A[6]*h_step*ff_x[i];
    Gs_x[i+4*N]     += A[7]*h_step*ff_x[i+1*N];
    Gs_x[i+4*N]     += A[8]*h_step*ff_x[i+2*N];
    Gs_x[i+4*N]     += A[9]*h_step*ff_x[i+3*N];
    Gs_y[i+4*N]     += A[6]*h_step*ff_y[i];
    Gs_y[i+4*N]     += A[7]*h_step*ff_y[i+1*N];
    Gs_y[i+4*N]     += A[8]*h_step*ff_y[i+2*N];
    Gs_y[i+4*N]     += A[9]*h_step*ff_y[i+3*N];
    Gs_theta[i+4*N] += A[6]*h_step*ff_theta[i];
    Gs_theta[i+4*N] += A[7]*h_step*ff_theta[i+1*N];
    Gs_theta[i+4*N] += A[8]*h_step*ff_theta[i+2*N];
    Gs_theta[i+4*N] += A[9]*h_step*ff_theta[i+3*N];
  }

  // Computing f(G5):
  compute_xx(t+C[4]*h_step, (Gs_x+4*N), (Gs_y+4*N), (Gs_theta+4*N), (ff_x+4*N), (ff_
y+4*N), (ff_theta+4*N));
}

void rk4::compute_y1y1h(double t, double* Gs_x, double* ff_x, double* Gs_y, double*
ff_y, double* Gs_theta, double* ff_theta){
  double sc_x[N];
  double sc_y[N];
  double sc_theta[N];
  for (int i=0; i<N; i++){
    sc_x[i] = 0.;
    sc_y[i] = 0.;
    sc_theta[i] = 0.;
    x1[i] = 0.;
    y1[i] = 0.;
    theta1[i] = 0.;
    x1h[i] = 0.;
    y1h[i] = 0.;
    theta1[i] = 0.;
```

```
  }

  for (int i=0; i<N; i++){
    x1[i]      = x0[i];
    y1[i]      = y0[i];
    theta1[i]  = theta0[i];
    x1h[i]     = x0[i];
    y1h[i]     = y0[i];
    theta1h[i] = theta0[i];
  }
  for (int i=0; i<N; i++){
    for (int j=0; j<4; j++){
      x1[i] += B[j]*h_step*ff_x[i+j*N];
      y1[i] += B[j]*h_step*ff_y[i+j*N];
      theta1[i] += B[j]*h_step*ff_theta[i+j*N];
    }
  }

  for (int i=0; i<N; i++){
    for (int j=0; j<5; j++){
      x1h[i] += B[j+4]*h_step*ff_x[i+j*N];
      y1h[i] += B[j+4]*h_step*ff_y[i+j*N];
      theta1h[i] += B[j+4]*h_step*ff_theta[i+j*N];
    }
  }
  for (int i=0; i<N; i++){
    sc_x[i] = Atol + Rtol * std::max(std::abs(x0[i]), std::abs(x1[i]));
    sc_y[i] = Atol + Rtol * std::max(std::abs(y0[i]), std::abs(y1[i]));
    sc_theta[i] = Atol + Rtol * std::max(std::abs(theta0[i]), std::abs(theta1[i]));
  }

  double err=0.0;

  for (int i=0; i<N; i++){
    err += ((x1[i]-x1h[i])/sc_x[i])*((x1[i]-x1h[i])/sc_x[i]);
    err += ((y1[i]-y1h[i])/sc_y[i])*((y1[i]-y1h[i])/sc_y[i]);
    err += ((theta1[i]-theta1h[i])/sc_theta[i])*((theta1[i]-theta1h[i])/sc_theta[i])
;
  }

  err *= (1./float(N));
  err = sqrt(err);
  last_h_step = h_step;
  h_step = h_step * std::min(facmax, std::max(facmin, fac*pow((1./err), (1./4.))));

//printf("Err=%f  -  h_step = %f\n", err, h_step);

  if (err>1){
    compute_Gs(t, Gs_x, ff_x, Gs_y, ff_y, Gs_theta, ff_theta);
    compute_y1y1h(t, Gs_x, ff_x, Gs_y, ff_y, Gs_theta, ff_theta);
  }
  else if (t+last_h_step+h_step>T_final){
    h_step = T_final-(t+last_h_step);
  }
}

void rk4::hermite(double actual_t, double myTheta, char* filenameDense){
  std::ofstream myDense;
  myDense.open(filenameDense);
  double f0_x[N];
  double f0_y[N];
  double f0_theta[N];
  double f1_x[N];
  double f1_y[N];
  double f1_theta[N];
  compute_xx(actual_t, x0, y0, theta0, f0_x, f0_y, f0_theta);
  compute_xx(actual_t + last_h_step, x1, y1, theta1, f1_x, f1_y, f1_theta);

  for(int i=0; i<N; i++){
    double u_x = (1-myTheta)*x0[i] + myTheta*x1[i] + myTheta*(myTheta-1)*((1-2*myThe
ta)*(x1[i]-x0[i]) + (myTheta-1)*last_h_step*f0_x[i]+myTheta*last_h_step*f1_x[i]);
    double u_y = (1-myTheta)*y0[i] + myTheta*y1[i] + myTheta*(myTheta-1)*((1-2*myThe
ta)*(y1[i]-y0[i]) + (myTheta-1)*last_h_step*f0_y[i]+myTheta*last_h_step*f1_y[i]);
```

```
    double u_theta = (1-myTheta)*theta0[i] + myTheta*theta1[i] + myTheta*(myTheta-1)
*((1-2*myTheta)*(theta1[i]-theta0[i]) + (myTheta-1)*last_h_step*f0_theta[i]+myTheta*
last_h_step*f1_theta[i]);
    myDense<<(actual_t + myTheta*last_h_step)<<" "<<u_x <<" "<< u_y<<" "<<u_theta<<s
td::endl;
  }
  myDense.close();
}

void rk4::dense_output(double t_){
  int m1=0;
  if (float(int(t_/dense_stpsze)) == t_/dense_stpsze){
    m1 = int(t_/dense_stpsze)-1;
  }
  else{
    m1 = int(t_/dense_stpsze);
  }
  int m2 = int((t_+last_h_step)/dense_stpsze);
  for (int k=(m1+1); k<(m2+1); k++){
    char filenameDense[32];
    sprintf(filenameDense, "Dense%04d.txt", k);
    hermite(t_, (dense_stpsze*k-t_)/last_h_step, filenameDense);
  }
}

void rk4::nextStep(){
  for(int i=0; i<N; i++){
    x0[i]=x1[i];
    y0[i]=y1[i];
    theta0[i]=theta1[i];
  }
}

void rk4::compute_solution(double T_final_){
  T_final = T_final_;
  dense_stpsze = double(T_final/double(n_intvls));
  double t=0;
  double Gs_x[5*N];
  double ff_x[5*N];
  double Gs_y[5*N];
  double ff_y[5*N];
  double Gs_theta[5*N];
  double ff_theta[5*N];
  initialize();
  while(t<T_final){
    compute_Gs(t, Gs_x, ff_x, Gs_y, ff_y, Gs_theta, ff_theta);
    compute_y1y1h(t, Gs_x, ff_x, Gs_y, ff_y, Gs_theta, ff_theta);
    dense_output(t);
    t += last_h_step;
    printf("t=%f, next_step=%f\n", t, h_step);
    step_counter += 1;
    nextStep();
  };
  printf("Done! It tool us %d steps to perform the entire integration.", step_counte
r);
}

void rk4::initialize(){
  srand (static_cast <unsigned> (time(0)));
  float max_xy = 2;
  float max_angle = 2*M_PI;
  printf("N=%d\n", N);
  for(int i=0; i<N; i++){
    float x;
    float y;
    float phse;
    do {
      x = static_cast <float> (rand()) / (static_cast <float> (RAND_MAX/max_xy));
      y = static_cast <float> (rand()) / (static_cast <float> (RAND_MAX/max_xy));
      phse = static_cast <float> (rand()) / (static_cast <float> (RAND_MAX/max_angle
));
      x-=1;
      y-=1;
```

```
    } while ((x*x + y*y)>1);
    x0[i] = x;
    y0[i] = y;
    theta0[i] = phse;
    vx0[i] = 0;
    vy0[i] = 0;
    omega0[i] = 0;
  }
}
```