**projectA2.cpp**

#include <iostream>

#include <fstream>

#include "fourvector.h"

#include "fourmatrix.h"

using namespace std;

//Program to evaluate damped/undamped double pendulum using RK4 method

int main(void)

{

double R; //mass ratio of lower to upper pendulum

cout << "Mass ratio R = ";

cin >> R;

double G; //damping constant

cout << "Damping constant G = ";

cin >> G;

double h; //time step-size

cout << "Time step-size dt = ";

cin >> h;

double t=0.0; //initialise time

double g=10.0; //gravitational acceleration

//set matrix operator

fourmatrix M(0.0, 0.0, -R-1.0, R+1.0, 0.0, 0.0, R, -R-1.0, 1.0, 0.0, -G, G\*(1.0-1.0/R), 0.0, 1.0, 0.0, -G/R);

//RK4 method

ofstream rk4results("rk4results.txt"); //output file for results

//initalise state vector (upper and lower angles)(upper and lower angular velocities)

fourvector v0(0.1, 0.0, 0.0, 0.0);

for (t=0.0; t<=100.01; t+=h) //time-range of computation

{

double a1 = v0.get(0); //upper angle

double a2 = v0.get(1); //lower angle

double a3 = v0.get(2); //upper angular velocity

double a4 = v0.get(3); //lower angular velocity

//calculate total energy of system

double E = g\*(1.0+R)\*a3\*a3/2.0 + g\*R\*a4\*a4/2.0 + g\*R\*a3\*a4 + g\*(1.0+R)\*a1\*a1/2.0 + R\*g\*a2\*a2/2.0;

//print results

rk4results << t << '\t' << //time

v0.get(0) << '\t' << v0.get(1) << '\t' << //angles

v0.get(2) << '\t' << v0.get(3) << '\t' << //angular velocities

E << endl; //total energy

fourvector k1 = (v0\*M)\*h;

fourvector v1 = v0 + k1\*0.5;

fourvector k2 = (v1\*M)\*h;

fourvector v2 = v0 + k2\*0.5;

fourvector k3 = (v2\*M)\*h;

fourvector v3 = v0 + k3;

fourvector k4 = (v3\*M)\*h;

v0 = v0 + (k1 + k2\*2.0 + k3\*2.0 + k4)\*(1.0/6.0); //RK4 iteration

}

return 0;

}

**fourmatrix.h**

//define fourmatrix class

#ifndef FOURMATRIX\_H

#define FOURMATRIX\_H

#include "fourvector.h"

class fourmatrix

{

private:

double matrix[16]; //array of matrix elements (column by column)

public:

//default constructor

fourmatrix()

{

for (int i=0;i<16;i++){

matrix[i]=0.0;

}

}

//constructor

fourmatrix(double a1, double a2, double a3, double a4,

double b1, double b2, double b3, double b4,

double c1, double c2, double c3, double c4,

double d1, double d2, double d3, double d4)

{

matrix[0]=a1; matrix[4]=b1; matrix[8]=c1; matrix[12]=d1;

matrix[1]=a2; matrix[5]=b2; matrix[9]=c2; matrix[13]=d2;

matrix[2]=a3; matrix[6]=b3; matrix[10]=c3; matrix[14]=d3;

matrix[3]=a4; matrix[7]=b4; matrix[11]=c4; matrix[15]=d4;

}

//Access method for matrix elements

double get (int i)

{

return matrix[i];

}

};

#endif

**fourvector.h**

//define fourvector class

#ifndef FOURVECTOR\_H

#define FOURVECTOR\_H

#include "fourmatrix.h"

class fourvector

{

private:

double vector[4]; //array of vector elements

public:

//default constructor

fourvector()

{

for (int i=0;i<4;i++){

vector[i]=0.0;

}

}

//constructor

fourvector(double a1, double a2, double a3, double a4) //a1,a2,a3,a4: 1st,2nd,3rd,4th row

{

vector[0]=a1;

vector[1]=a2;

vector[2]=a3;

vector[3]=a4;

}

//Access method for vector elements

double get (int i)

{

return vector[i];

}

//Modifier method for vector elements

void set(double a1, double a2, double a3, double a4)

{

vector[0]=a1;

vector[1]=a2;

vector[2]=a3;

vector[3]=a4;

}

//Overloading \* operator as scalar multiplication

fourvector operator\*(double factor)

{

fourvector product(vector[0]\*factor,vector[1]\*factor,vector[2]\*factor,vector[3]\*factor);

return product;

}

//Overloading \* operator as multiplying fourmatrix

fourvector operator\*(fourmatrix T)

{

double a1 = vector[0]\*T.get(0) + vector[1]\*T.get(4) + vector[2]\*T.get(8) + vector[3]\*T.get(12);

double a2 = vector[0]\*T.get(1) + vector[1]\*T.get(5) + vector[2]\*T.get(9) + vector[3]\*T.get(13);

double a3 = vector[0]\*T.get(2) + vector[1]\*T.get(6) + vector[2]\*T.get(10) + vector[3]\*T.get(14);

double a4 = vector[0]\*T.get(3) + vector[1]\*T.get(7) + vector[2]\*T.get(11) + vector[3]\*T.get(15);

fourvector product (a1, a2, a3, a4);

return product;

}

//Overloading + operator

fourvector operator+(fourvector v\_other)

{

fourvector v\_sum(v\_other.get(0) + vector[0],

v\_other.get(1) + vector[1],

v\_other.get(2) + vector[2],

v\_other.get(3) + vector[3]);

return v\_sum;

}

};

#endif