Homework 5: ODEs

Fis. Computacional 2022-23 (P4)

Fernando Barão

12 de junho de 2023

ODE's (ordinary differential equations)

pêndulo gravítico

Consideremos um pêndulo simples gravítico sem presença de força de atrito, com um fio de comprimento $L = 4$ metros e uma massa $m = 500$ gramas na extremidade.

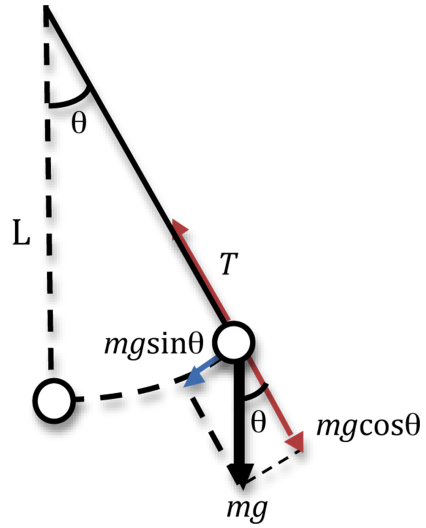

Figura 1: pêndulo simples

1. Determine a equação do movimento do pêndulo.
2. A equação do movimento do pêndulo é uma equação de segunda ordem (ordem da maior derivada) na coordenada angular (θ). Reduza esta equação a um sistema de duas equações diferenciais de primeira ordem.
3. Resolva numericamente a equação do movimento com os seguintes métodos:

   a) Trapezoidal
   b) Runge-Kutta de 2a e 4a ordem

   Para a resolução numérica:

   - implemente as classes **Xvar**, **ODEpoint** e **ODEsolver**.
     As primeiras duas classes (**Xvar** e **ODEpoint**) armazenam as variáveis independente (tempo) e dependentes (posição e velocidade).
     A classe **ODEsolver** deve possuir os diferentes métodos de resolução do problema.
   - construa um programa principal num ficheiro **main/rPendulum.{C,cpp}**

declaração da classe Xvar (ficheiro: src/ODEpoint.h)

```
class Xvar {
public:
Xvar() = default;
Xvar(int); // number of dependent variables
```

```
Xvar(std::vector<double>); // passing vector
// using initializer list to build object: X({1,2})
Xvar(const std::initializer_list<double>& v);
~Xvar();

Xvar(const X&); // copy constructor
Xvar& operator=(const Xvar&); // assignment operator
Xvar operator+(const Xvar&); // operator+
double& operator[](int);  // X[i]

friend Xvar operator*(double, const Xvar&); // scalar*X
friend std::ostream& operator<< (std::ostream&, const Xvar&);

std::vector<double>& X(); // accessor to x

protected:
std::vector<double> x;
};
```

declaração da classe ODEpoint (ficheiro: src/ODEpoint.h)

```
class ODEpoint : public Xvar {
private:
double t; // time

public:
ODEpoint() : t(-1) {;}
ODEpoint(double t_, Xvar a_) : t(t_), Xvar(a_) {;}
ODEpoint(double t_, const std::vector<double>& v) : t(t_), Xvar(v)
 ↪  {;}
ODEpoint(double t_, const std::initializer_list<double>& v) : t(t_),
 ↪  Xvar(v) {;}

void SetODEpoint(double t_, Xvar& p);
void SetODEpoint(double t_, const std::initializer_list<double>& v);
void SetODEpoint(double t_, std::vector<double> v);

double& T() { return t;} // accessor to time
};
```

declaração da classe ODEsolver (ficheiro: src/ODEsolver.h)

```
class ODEsolver {
public:

 ODEsolver(const vector<std::function<double(ODEpoint)>>&);
 ~ODEsolver();

 // set functions
 SetODEfunc(const vector<std::function<double(ODEpoint)>>&);

 // solver methods
 const vector<ODEpoint>& Euler(ODEpoint i, double step, double T);
 const vector<ODEpoint>& PredictorCorrector(ODEpoint i, double step,
  ↪  double T);
 const vector<ODEpoint>& LeapFrog(ODEpoint i, double step, double T);
 const vector<ODEpoint>& RK2(ODEpoint i, double step, double T);
 const vector<ODEpoint>& RK4(ODEpoint i, double step, double T);
 (...)

private:
 vector<std::function<double(ODEpoint)>> F;
 map<string,  vector<ODEpoint> > MS; // key: "euler", "trapezoidal",
 ↪  ...
 (...)
};
```