Comp. Methods in Mech. Eng.

MCG 4127

Assignment # 6

uOttawa

Name: Usama Tariq

Student Number: 7362757

Professor: Catherine Mavriplis

TA: Fabien Giroux

Due Date: February 26, 2019

In this assignment the problem was to model the height of water emptying out a cone shaped container with respect to time. Time stepping methods such as Forward Euler and Runge-Kutta are used and compared to each other and the analytical solution.

From fluid dynamics the conservation of mass states the following:

$$0 = \frac{\partial}{\partial t}\iiint \rho dV + \oiint \rho \vec{v}\, dA$$

From the conservation of mass equation, we can derive an ordinary differential equation which models the height with respect to time as written below:

$$\frac{dh}{dt} = -\frac{\sqrt{2g}}{\pi\alpha^2} A_{exit} h^{-\frac{3}{2}}$$

This ODE is solvable through separation of variables and has the following analytical solution:

$$h = \sqrt[\frac{2}{5}]{-\frac{5}{2}\frac{\sqrt{2g}}{\pi\alpha^2} A_{exit} t + 20^{\frac{5}{2}}}$$

However, the goal in this assignment was to use numerical methods via Euler and Runge-Kutta.

Euler's Method

Using Euler's Forward method, which is represented by:

$$h^{(n+1)} = h^{(n)} + \frac{dh}{dt}\Delta t$$

The following graphs in Figure 1 and Figure 2 are obtained. The code for which can be seen in Appendix A

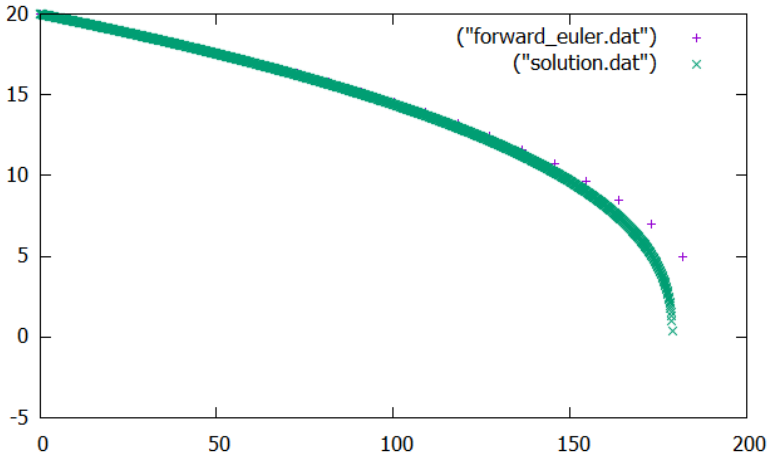


Figure 1 Forward Euler Method Using a Larger Time Step

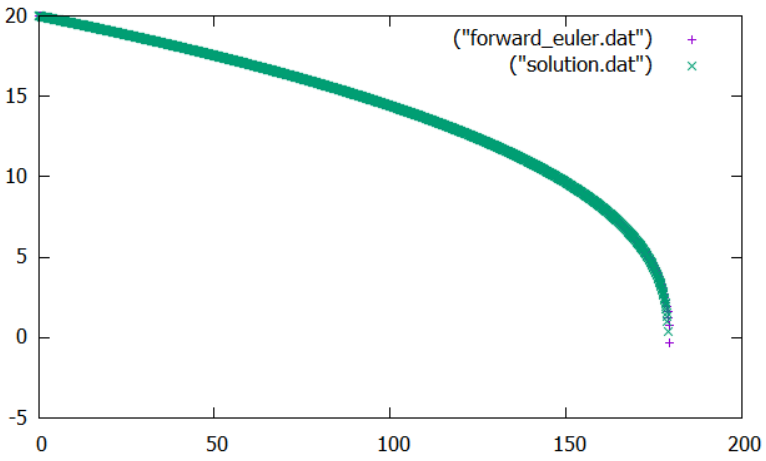


Figure 2 Forward Euler Method Using a Smaller Time Step

As can be seen in Figure 1 and 2 when the time step decreases, the resolution gets better.

Runge-Kutta

Runge-Kutta is represented by the following equation:

$$h^{(n+1)} = h^{(n)} + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}\Delta t$$

Where:

$$k_1 = \frac{dh}{dt}$$

$$k_2 = \frac{d\left(h^{(n)} + \frac{k_1 \Delta t}{2}\right)}{dt}$$

$$k_3 = \frac{d\left(h^{(n)} + \frac{k_2 \Delta t}{2}\right)}{dt}$$

$$k_4 = \frac{d\left(h^{(n)} + k_3 \Delta t\right)}{dt}$$

When this method is coded (See Appendix A) the following graphs are obtained (seen in Figure 3 and 4):

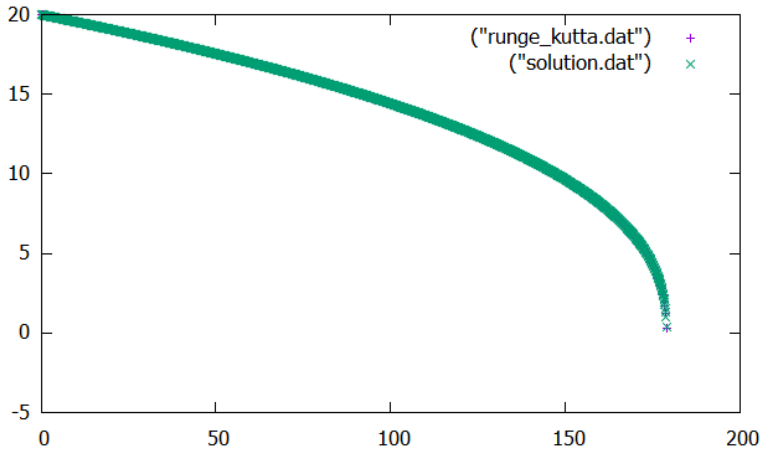


Figure 3 Runge-Kutta Using a Larger Time Step

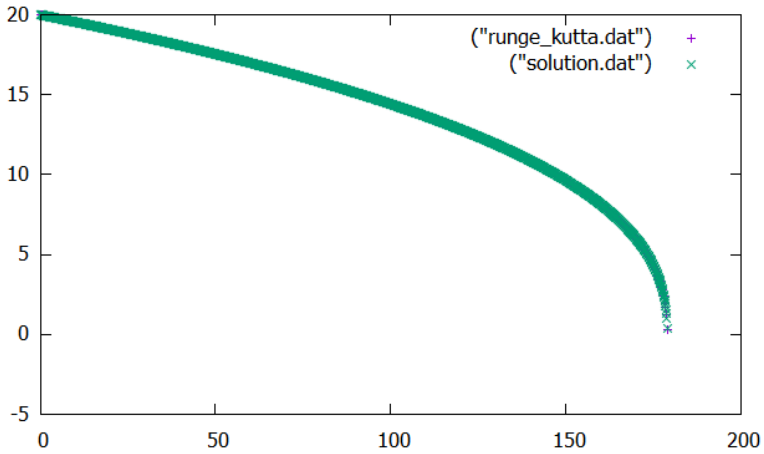
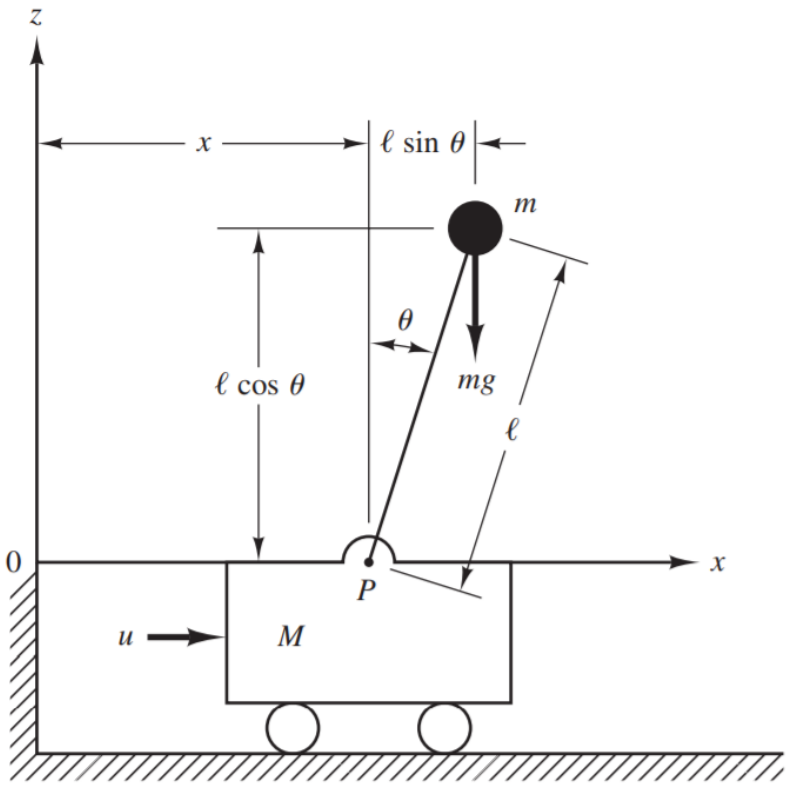


Figure 4 Runge-Kutta using a smaller Time Step

From the two figures shown above, Runge-Kutta is a far better method than Euler as it requires the time steps to be larger than Euler to produce the same result. The following table summarizes two points on the curve using a few different delta t (time steps) to show convergence.

$\Delta t(s)$	**Forward Euler**	**Error (%)**	**Runge-Kutta**	**Error (%)**	**Solution**
9.1	15.1176	0.437157018	15.0518	0	15.0518
3.64	15.1038	0.345473631	15.0518	0	15.0518
0.182	15.0545	0.017938054	15.0518	0	15.0518
0.091	15.0531	0.008636841	15.0518	0	15.0518

Inverted Pendulum



In this part of the assignment an inverted pendulum was to be modelled that was moving on a block, having two degrees of freedom.

The differential equations that needed to be solved was:

$$x_1 = \theta$$

$$x_2 = \dot{\theta}$$

$$x_3 = x$$

$$x_4 = \dot{x}$$

Using this information, the following ODE's can be analyzed:

$$\dot{x_1} = x_2$$

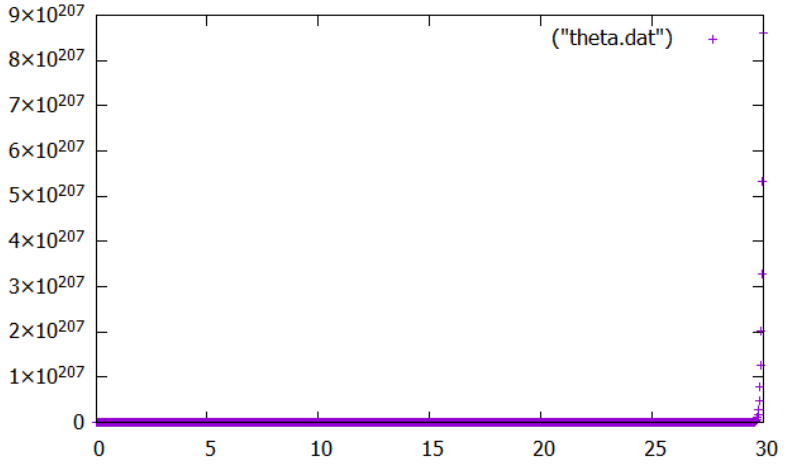
$$\dot{x_2} = \frac{M+m}{Ml} g x_1 - \frac{1}{Ml} u$$

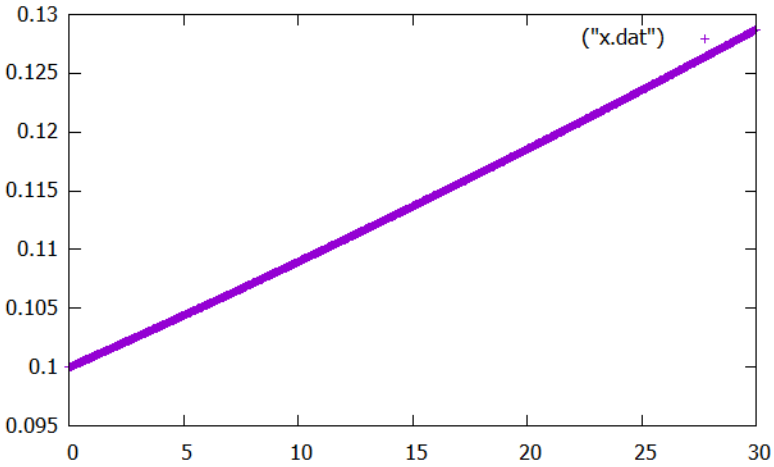
$$\dot{x_3} = x_4$$

$$\dot{x_4} = -\frac{m}{M} g x_1 + \frac{1}{M} u$$

Where $u = \sin(x)$

An appropriate initial condition for the system was $theta = 0.1$, $theta_{dot} = 0.1, x = 0.1$ and $x_{dot} = 0.1$, which when plotted resulted in Figure 5 and 6.





It can be seen from the figures above that the system was unstable.

Appendix A

```
1.  #include <vector>
2.  #include <cmath>
3.  #include <stdexcept>
4.  #include <iostream>
5.  #include <fstream>
6.  #include <functional>
7.
8.  using function_type = std::function<double(double)>;
9.
10. void output_solution(function_type solution, const std::string filename){
11.   std::ofstream fout(filename);
12.   if(!fout){
13.     throw std::runtime_error("Could not open file: " + filename);
14.   }
15.
16.   for (double t = 0; t < 180; t = t + 0.1){
17.     fout <<  t << " " << solution(t) << '\n';
18.     std::cout << t << " " << solution(t) << '\n';
19.   }
20. }
21.
22. void output_to_file(const std::vector<double> ts, const std::vector<double> ys, const s
    td::string filename){
23.   if (ts.size() != ys.size()){
24.     throw std::runtime_error("Vectors have different sizes");
25.   }
26.
27.   std::ofstream fout(filename);
28.   if (!fout){
29.     throw std::runtime_error("Could not open file: " + filename);
30.   }
31.
32.   for (int i = 0; i < ts.size(); ++i){
33.     fout << std::setw(30) << ts[i] << std::setw(30) << ys[i] << '\n';
34.     std::cout << std::setw(30) << ts[i] << std::setw(30) << ys[i] << '\n';
35.   }
36. }
37.
38. auto forward_euler(double h0, double t, double h, double delta_t, int num, function_typ
    e ode){
39.
40.   std::vector<double> times({0.0});
41.   std::vector<double> hs({h0});
42.
43.   for (int i = 0; i < num; i++){
44.     h = h + delta_t * ode(h);
45.     t += delta_t;
46.     times.push_back(t);
47.     hs.push_back(h);
48.   }
49.
50.   return std::make_pair(times, hs);
51. }
52.
53. auto runge_kutta (double h0, double t, double h, double delta_t, int num, function_type
     ode){
54.
55.   std::vector<double> times({0.0});
```

```cpp
  std::vector<double> hs({h0});
  double k1, k2, k3, k4;

  for (int i = 0; i < num; i++){
    k1 = ode(h);
    k2 = ode(h + k1*delta_t/2);
    k3 = ode(h + k2*delta_t/2);
    k4 = ode(h + k3*delta_t);

    h = h + (delta_t*(k1 + 2*k2 + 2*k3 + k4))/6;
    t += delta_t;
    times.push_back(t);
    hs.push_back(h);
  }

  return std::make_pair(times, hs);
}

void pendulum(){
  double m1 = 2;
  double m2 = 0.1;
  double l = 0.5;
  double g = 9.81;
  double final_time = 30;
  int num = 1000;


  auto ode_theta = [](const double theta){
    return (2+0.1)*9.81*theta/(2*0.5) - sin(theta)/(2*0.5);
  };

  auto ode_x = [](const double x){
    return -0.1*9.81*x/2 + sin(x)/2;
  };

  double theta0 = 0.1;
  double x0 = 0.1;
  double t = 0.0;
  double theta = theta0;
  double x = x0;

  double delta_t = final_time/static_cast<double>(num);

        auto theta_result = forward_euler(theta0, t, theta, delta_t, num, ode_theta);

        std::cout << "-------------------------------------------------------\n";
        std::cout << "-------------------------------------------------------\n";
        std::cout << "|                   THETA                             |\n";
        std::cout << "-------------------------------------------------------\n";
        std::cout << "-------------------------------------------------------\n";
        output_to_file(theta_result.first, theta_result.second, "theta.dat");
        auto x_result = forward_euler(x0, t, x, delta_t, num, ode_x);
        std::cout << "-------------------------------------------------------\n";
        std::cout << "-------------------------------------------------------\n";
        std::cout << "|                   X                                 |\n";
        std::cout << "-------------------------------------------------------\n";
        std::cout << "-------------------------------------------------------\n";
        output_to_file(x_result.first, x_result.second, "x.dat");
      }

```

```
116.
117.       int main(){
118.
119.         double final_time = 182;
120.         int num = 50;
121.
122.
123.         auto ode = [](const double h) {
124.           return (-sqrt(2*32.3)*0.25*pow(h, -1.5)) / (3.14*pow(0.4,2));
125.         };
126.
127.         auto solution = [](const double t) {
128.           return pow(-
    2.5*sqrt(2*32.3)*0.25*t/(3.14*pow(0.4,2)) + pow(20, 2.5) , 0.4);
129.         };
130.
131.         double h0 = solution(0.0);
132.         double t = 0.0;
133.         double h = h0;
134.         double delta_t = final_time/static_cast<double>(num);
135.
136.         auto forward_euler_result = forward_euler(h0, t, h, delta_t, num, ode);
137.         auto runge_kutta_result = runge_kutta(h0, t, h, delta_t, num, ode);
138.
139.         std::cout << "-----------------------------------------------------\n";
140.         std::cout << "-----------------------------------------------------\n";
141.         std::cout << "|                    FORWARD EULER                  |\n";
142.         std::cout << "-----------------------------------------------------\n";
143.         std::cout << "-----------------------------------------------------\n";
144.         output_to_file(forward_euler_result.first,forward_euler_result.second, "forwar
    d_euler.dat");
145.
146.         std::cout << "-----------------------------------------------------\n";
147.         std::cout << "-----------------------------------------------------\n";
148.         std::cout << "|                     RUNGE KUTTA                   |\n";
149.         std::cout << "-----------------------------------------------------\n";
150.         std::cout << "-----------------------------------------------------\n";
151.         output_to_file(runge_kutta_result.first, runge_kutta_result.second, "runge_kut
    ta.dat");
152.
153.         std::cout << "-----------------------------------------------------\n";
154.         std::cout << "-----------------------------------------------------\n";
155.         std::cout << "|                     SOLUTION                      |\n";
156.         std::cout << "-----------------------------------------------------\n";
157.         std::cout << "-----------------------------------------------------\n";
158.         output_solution(solution, "solution.dat");
159.
160.         pendulum();
161.
162.       }
```