APSC Exercise Session

April 17, 2015

Exercise

Consider the code below that solves the ODE

$$\dot{x} = -kx$$

```
#include <iostream>
#include <vector>
#include <cmath>

typedef double real;

int main (void)
{

  real start = 0;
  real stop = 10;
  real dt0 = .1;
  real tol = 1e-3;
  real x0 = 10;

  std::vector<double> result;
  std::vector<double> time;

  real t = start;
  real dt = dt0;
  real x = x0;
  real k = .5;
  real xold = x;
  real x2 = x;
  real err = 10 * tol;
  real told = t;
  real xprime = 0;

  result.clear ();
  time.clear ();

  result.push_back (x);
```

```
  time.push_back (t);

  while (t < stop)
    {

      xold = x;
      told = t;
      xprime = -k * x;

        t += dt;
        x += xprime * dt;
        xprime = - k * x;

        real x2 = .5 * (x + xold + xprime * dt);
        real err = fabs (x2 - x);
        if (err <= tol)
          {
            result.push_back (x);
            time.push_back (t);
          }
        else
          {
            std::cout << "%% reject" << std::endl;
            x = xold;
            t = told;
          }

        std::cout << "% t = "
                  << t << ";\t x = "
                  << x << std::endl;

        dt *= .5 * sqrt (tol / err);
        dt = t + dt > stop ? stop - t : dt;
    }

  std::cout << "r = [" << std::endl;
  for (auto ii = x.begin (),  jj = t.begin ();
       ii != x.end () || jj != t.end ();
       ++ii, ++jj)
    {
      std::cout << *jj << ",\t" << *ii << std::endl;
    }

  std::cout << "];" << std::endl;

  return 0;
}
```

1. Refactor the code creating a class `forward_euler` that has a constructor to set up method

parameters and a method `forward_euler::apply ()` to solve the equation. Let the derivative function be a functor that is passed to the `forward_euler::apply ()` method.

2. Refactor the code creating a class `forward_euler` that has a constructor to set up method parameters and a method `forward_euler::apply ()` to solve the equation. Now define the derivative $\dot{x}$ as a function taking 3 input arguments `real func (real x, real t, real k)` and use `C++11 lambdas` to pass it to `forward_euler::apply ()`

3. Modify the code from previous points to use a formal argument of type `std::function` instead of a template parameter

Solution

1. Solution to question 1:

```
#include <iostream>
#include <vector>
#include <cmath>

template<typename real>
class
forward_euler
{
public:

  forward_euler (real _start, real _stop,
                 real _dt0, real _tol) :
    start (_start), stop (_stop),
    dt0 (_dt0), tol (_tol)
  {};

  template<class T>
  void apply (T fun, real x0,
         std::vector<real>& result,
         std::vector<real>& time)
  {
    real t = start;
    real dt = dt0;
    real x = x0;
    real xold = x;
    real x2 = x;
    real err = 10 * tol;
    real told = t;
    real xprime = 0;

    result.clear ();
    time.clear ();

    result.push_back (x);
    time.push_back (t);
```

```
    while (t < stop)
      {

        xold = x;
        told = t;
        xprime = fun (x, t);

        t += dt;
        x += xprime * dt;
        xprime = fun (x, t);

        real x2 = .5 * (x + xold + xprime * dt);
        real err = fabs (x2 - x);
        if (err <= tol)
          {
            result.push_back (x);
            time.push_back (t);
          }
        else
          {
            std::cout << "%% reject" << std::endl;
            x = xold;
            t = told;
          }

        std::cout << "% t = " << t
                  << ";\t x = " << x
                  << std::endl;

        dt *= .5 * sqrt (tol / err);
        dt = t + dt > stop ? stop - t : dt;
      }
  }

private:

  real start, stop, dt0, tol;

};

template<typename real>
class fun
{
private:
  real k;
public:

  fun (real _k): k(_k) {};
```

```
  real operator() (real x, real t) {return (-k * x); };
};

int main (void)
{
  std::vector<double> x;
  std::vector<double> t;

  fun<double> fcn (.5);
  forward_euler<double> f (0.0, 100.0, .1, 1e-3);
  f.apply (fcn, 10, x, t);

  std::cout << "r = [" << std::endl;
  for (auto ii = x.begin (),  jj = t.begin ();
       ii != x.end () || jj != t.end ();
       ++ii, ++jj)
    {
      std::cout << *jj << ",\t " << *ii << std::endl;
    }

  std::cout << "];" << std::endl;

  return 0;
}
```

2. Solution to question 2:

```
#include <iostream>
#include <vector>
#include <cmath>

template<typename real>
class
forward_euler
{
public:

  forward_euler (real _start, real _stop,
                 real _dt0, real _tol) :
    start (_start), stop (_stop),
    dt0 (_dt0), tol (_tol)
  {};

  template<class T>
  void apply (T fun, real x0,
         std::vector<real>& result,
         std::vector<real>& time)
  {
    real t = start;
```

```
    real dt = dt0;
    real x = x0;
    real xold = x;
    real x2 = x;
    real err = 10 * tol;
    real told = t;
    real xprime = 0;

    result.clear ();
    time.clear ();

    result.push_back (x);
    time.push_back (t);

    while (t < stop)
      {

        xold = x;
        told = t;
        xprime = fun (x, t);

        t += dt;
        x += xprime * dt;
        xprime = fun (x, t);

        real x2 = .5 * (x + xold + xprime * dt);
        real err = fabs (x2 - x);
        if (err <= tol)
          {
            result.push_back (x);
            time.push_back (t);
          }
        else
          {
            std::cout << "%% reject" << std::endl;
            x = xold;
            t = told;
          }

        std::cout << "% t = " << t
                  << "; \t x = " << x
                  << std::endl;

        dt *= .5 * sqrt (tol / err);
        dt = t + dt > stop ? stop - t : dt;
      }
  }

private:
```

```
  real start, stop, dt0, tol;

};

template<typename real>
real fun (real x, real t, real k)
{
  return (-sin (k * x));
};

int main (void)
{
  std::vector<double> x;
  std::vector<double> t;

  forward_euler<double> f (0.0, 100.0, .1, 1e-3);
  f.apply ([] (double xx, double tt)
           {return fun<double> (xx, tt, 1.0 / 2.0);},
           10, x, t);

  std::cout << "r = [" << std::endl;
  for (auto ii = x.begin (), jj = t.begin ();
       ii != x.end () || jj != t.end ();
       ++ii, ++jj)
    {
      std::cout << *jj << ",\t " << *ii << std::endl;
    }

  std::cout << "];" << std::endl;

  return 0;
}
```