NUM2cv
Vladislav Belov

Řešíme numericky Riccatiho rovnici (1) na intervalu $[0.25, 0.45]$ Rungeovými-Kuttovými metodami. Známe její analytické řešení: $u(t) = (\frac{1}{\sqrt{2}t}\tan(\sqrt{2}(c - \frac{1}{t})) - \frac{1}{2t})e^t$, kde klademe $c = 1$.

$$\begin{aligned} \dot{u}(t) &= t^{-4}e^t + u(t) + 2e^{-t}u^2(t) = f(t,u), \\ u(0.25) &= -31,1844. \end{aligned} \tag{1}$$

Označíme $\tau = \mathit{integrationTimeStep}$, $u_0 = u(t_0)$. Použijeme metody:

I. Euler:

$$\begin{aligned} k_1(\tau) &= \tau \cdot f(t_0, u_0), \\ u(t_0 + \tau) &= u(t) + k_1(\tau). \end{aligned}$$

II. Runge-Kutta 2. řádu:

$$\begin{aligned} k_1(\tau) &= \tau \cdot f(t_0, u_0), \\ k_2(\tau) &= \tau \cdot f(t_0 + \tau, u_0 + k_1(\tau)), \\ u(t_0 + \tau) &= u(t) + \tfrac{1}{2}k_1(\tau) + \tfrac{1}{2}k_2(\tau). \end{aligned}$$

III. Runge–Kutta–Merson:

$$\begin{aligned} k_1(\tau) &= \tau \cdot f(t_0, u_0), \\ k_2(\tau) &= \tau \cdot f(t_0 + \tfrac{1}{3}\tau, u_0 + \tfrac{1}{3}k_1), \\ k_3(\tau) &= \tau \cdot f(t_0 + \tfrac{1}{3}\tau, u_0 + \tfrac{1}{6}k_1 + \tfrac{1}{6}k_2), \\ k_4(\tau) &= \tau \cdot f(t_0 + \tfrac{1}{2}\tau, u_0 + \tfrac{1}{8}k_1 + \tfrac{3}{8}k_3), \\ k_5(\tau) &= \tau \cdot f(t_0 + \tau, u_0 + \tfrac{1}{2}k_1 - \tfrac{3}{2}k_3 + 2k_4), \\ u(t_0 + \tau) &= y_0 + \tfrac{1}{6}k_1 + \tfrac{2}{3}k_4 + \tfrac{1}{6}k_5. \end{aligned}$$

Rozebereme podrobněji metodu (II):

(a) Výsledný graf v porovnání s průběhem analytického řešení:

(b) Implementace:

```
template< typename Problem >
class RungeKutta : public IntegratorBase
{
 public:

  RungeKutta( Problem& problem )
  {
       this->k1 = new double[ problem.getDegreesOfFreedom() ];
       this->k2 = new double[ problem.getDegreesOfFreedom() ];
       this->aux = new double[ problem.getDegreesOfFreedom() ];
  }

  bool solve( Problem& problem,
              double* u )
  {
       const int dofs = problem.getDegreesOfFreedom();
       double tau = std::min( this->integrationTimeStep, this->stopTime - this->time );
       long int iteration( 0 );
       while( this->time < this->stopTime )
       {
         /****
         * Compute k1
         */
         problem.getRightHandSide( this->time, u, k1 );
```

```
    /****
     * Compute k2
     */
    for( int i = 0; i < dofs; i++ )
      aux[ i ] = u[ i ] + tau * k1[ i ];
    problem.getRightHandSide( this->time + tau, aux, k2 );

    /****/
    for( int i = 0; i < dofs; i++ )
      u[ i ] += ( tau / 2.0 ) * ( k1[ i ] + k2[ i ] );
    this->time += tau;
    iteration++;
    if( iteration > 100000 )
    {
      std::cerr << "The solver has reached the maximum number of iteratoins. "
                << std::endl;
      return false;
    }
    tau = std::min( tau, this->stopTime - this->time );
    std::cout << "ITER: " << iteration << " \t tau = " << tau
              << " \t time= " << time << "          \r " << std::flush;
  }
  std::cout << std::endl;
  return true;
}

~RungeKutta()
{
  delete[] k1;
  delete[] k2;
  delete[] aux;
}

protected:

double *k1, *k2, *aux;

};
```