Software Lab CES

Type-Generic Solution for Systems of Ordinary Differential Equations

Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen University

Contents

Outline

Objective

- Introduction to design and implementation of a type-generic solution infrastructure for parameterized systems of explicit ordinary differential equations

Learning Outcomes

- You will understand
  - requirements, design, implementation of the sample code
- You will be able to
  - download, build and run the sample code.

Outline

We consider the solution $\mathbf{x}(T, \mathbf{p}),\ T > 0$ of an initial value problem for systems of parameterized explicit ordinary differential equations (ODE)

$$\frac{d\mathbf{x}}{dt} = g(\mathbf{x}(t, \mathbf{p}), \mathbf{p}), \quad \mathbf{x}(0, \mathbf{p}) = \mathbf{x}^0 \ .$$

Forward finite differences in time with appropriate fixed time step $0 < \Delta t \ll 1$ yield

$$\frac{\mathbf{x}(t + \Delta t, \mathbf{p}) - \mathbf{x}(t, \mathbf{p})}{\Delta t} \approx g(\mathbf{x}(t, \mathbf{p}), \mathbf{p})$$

and, hence, $\mathbf{x} = \mathbf{x}(T, \mathbf{p})$ as result of $n_t = \frac{T}{\Delta t}$ explicit Euler steps of the form

$$\mathbf{x}(t + \Delta t, \mathbf{p}) = \mathbf{x}(t, \mathbf{p}) + \Delta t \cdot g(\mathbf{x}(t, \mathbf{p}), \mathbf{p})$$
$$t = t + \Delta t$$

starting from $t = 0$.

Backward finite differences in time with appropriate fixed time step $0 < \Delta t \ll 1$ yield

$$\frac{\mathbf{x}(t, \mathbf{p}) - \mathbf{x}(t - \Delta t, \mathbf{p})}{\Delta t} \approx g(\mathbf{x}(t, \mathbf{p}), \mathbf{p})$$

and, hence, $\mathbf{x} = \mathbf{x}(t, \mathbf{p})$ as solution of the system of nonlinear equations

$$f(\mathbf{x}, \mathbf{p}, t, \Delta t) = \mathbf{x} - \mathbf{x}(t - \Delta t, \mathbf{p}) - \Delta t \cdot g(\mathbf{x}, \mathbf{p}) = 0 \, .$$

Starting from $\mathbf{x} = \mathbf{x}^0$ at $t = 0$ this implicit Euler method performs $n_t = \frac{T}{\Delta t}$ time steps as

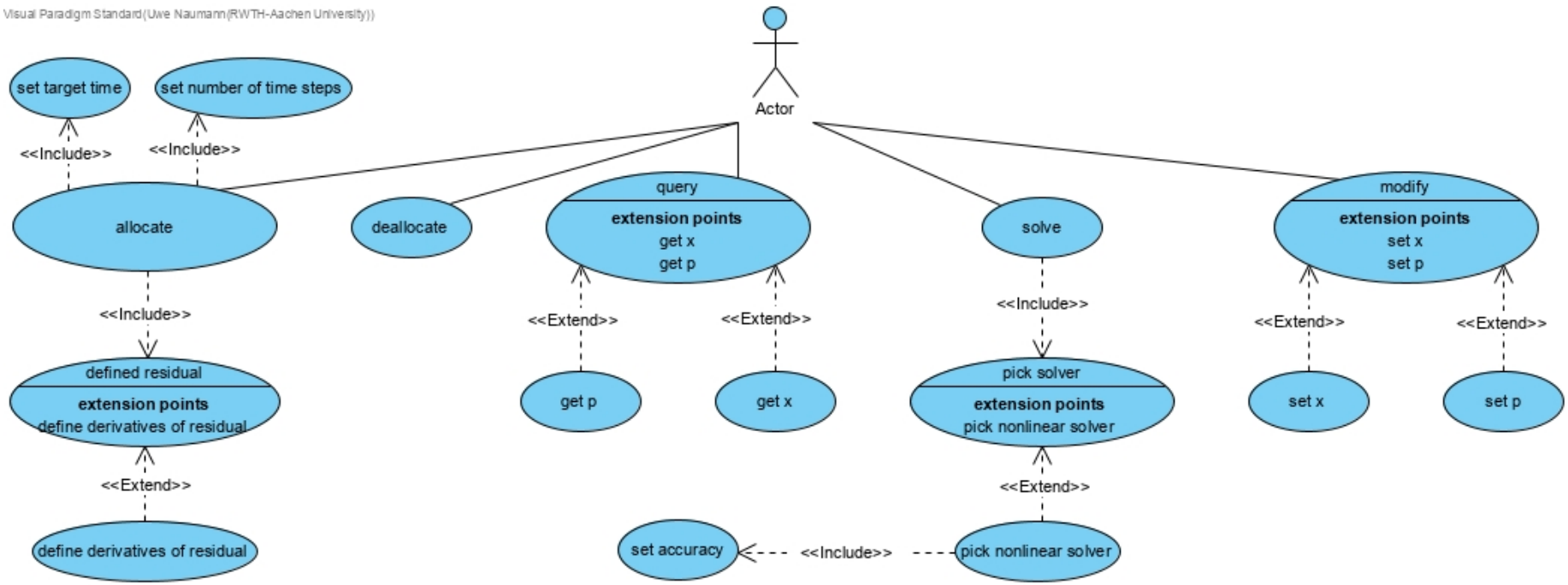
$$\mathbf{x}(t, \mathbf{p}) - \mathbf{x}(t - \Delta t, \mathbf{p}) - \Delta t \cdot g(\mathbf{x}(t, \mathbf{p}), \mathbf{p}) = 0 \quad [\Rightarrow \mathbf{x}(t, \mathbf{p})]$$
$$t = t + \Delta t \, .$$

Outline

I am looking for a software library for solving parameterized systems of explicit ordinary differential equations $\frac{d\mathbf{x}}{dt} = g(\mathbf{x}(t, \mathbf{p}), \mathbf{p}) = 0$ including

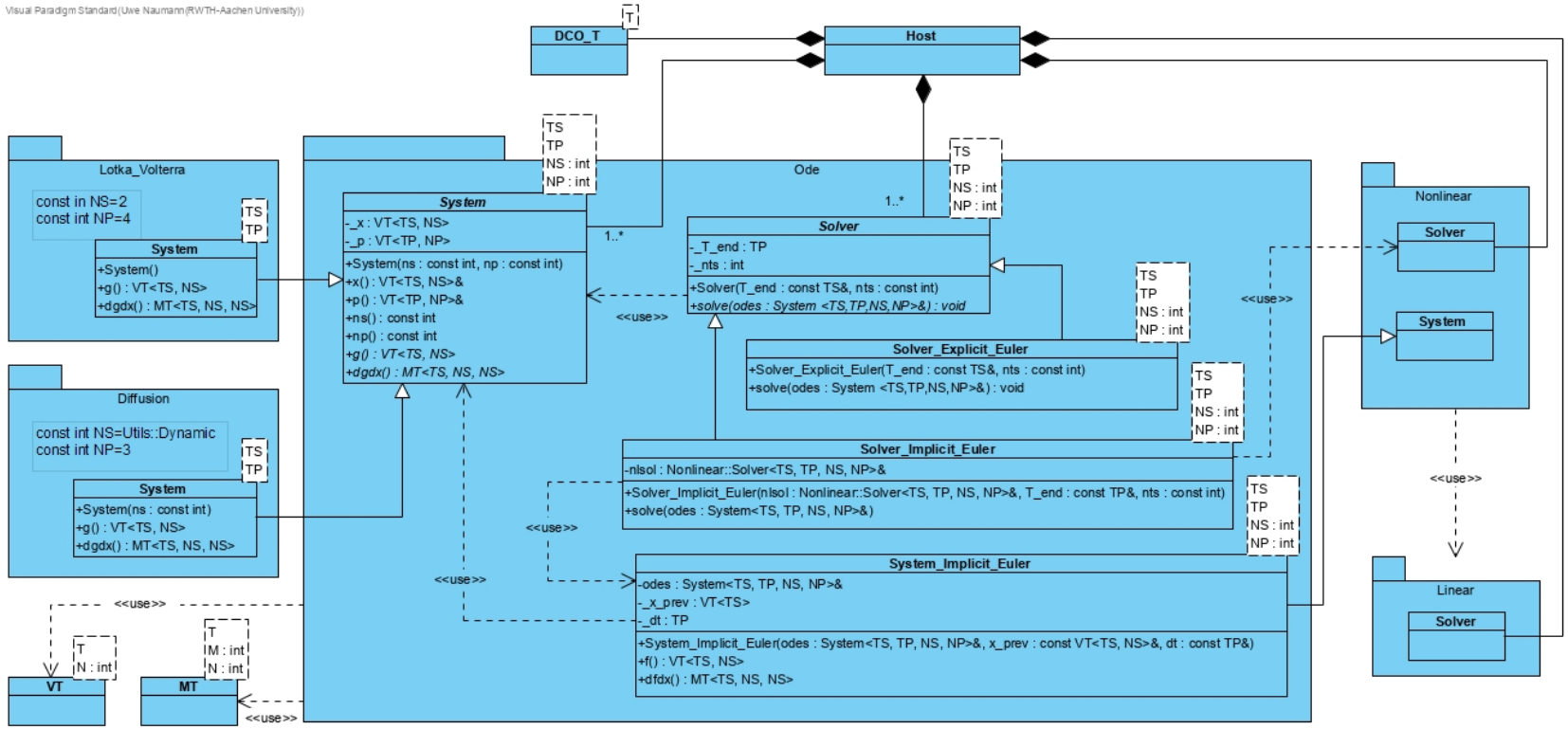
- definition of the residual $g : \boldsymbol{R}^{n_s} \times \boldsymbol{R}^{n_p} \to \boldsymbol{R}^{n_s}$ as well as of potentially required derivatives thereof
- definition of the target time $T \in \boldsymbol{R}$ and of the number of time steps $nt > 0$ to be performed by the integrator
- definition, storage and extraction of the state $\mathbf{x} \in \boldsymbol{R}^{n_s}$ and of the parameters $\mathbf{p} \in \boldsymbol{R}^{n_p}$
- demonstrated extensible choice of integrators (explicit and implicit Euler to get started)

Algorithmic differentiation with dco/c++ shall be used for sensitivity analysis of functions of $\mathbf{x}$ (e.g, $\|\mathbf{x}\|_2$) wrt. to perturbations in $\mathbf{p}$.


Visual Paradigm Standard(Uwe Naumann(RWTH-Aachen University))
set target time
set number of time steps
Actor
<<Include>>
<<Include>>
allocate
deallocate
query
extension points
get x
get p
solve
modify
extension points
set x
set p
<<Include>>
<<Extend>>
<<Extend>>
<<Include>>
<<Extend>>
<<Extend>>
defined residual
extension points
define derivatives of residual
get p
get x
pick solver
extension points
pick nonlinear solver
set x
set p
<<Extend>>
<<Extend>>
define derivatives of residual
set accuracy
<<Include>>
pick nonlinear solver

- ODE system
  - allocation including definition of target time and of number of time steps
  - deallocation
  - generic element types TS of elements of state and TP of parameters enabling overloading and corresponding matrix and vector types.
  - read/write access routines for **x** and **p**
  - implementation of residual and of required derivatives
  - nonlinear system in case of implicit Euler (single Newton step in linear case)
- ODE solver
  - allocation
  - deallocation
  - solution of ODE system using nlslib as required
  - abstraction for extensibility
  - overloading

Outline


Visual Paradigm Standard(Uwe Naumann(RWTH-Aachen University))
DCO_T
T
Host
Lotka_Volterra
const in NS=2
const int NP=4
TS
TP
System
+System()
+g() : VT<TS, NS>
+dgdx() : MT<TS, NS, NS>
Ode
System
-_x : VT<TS, NS>
-_p : VT<TP, NP>
+System(ns : const int, np : const int)
+x() : VT<TS, NS>&
+p() : VT<TP, NP>&
+ns() : const int
+np() : const int
+g() : VT<TS, NS>
+dgdx() : MT<TS, NS, NS>
TS
TP
NS : int
NP : int
1..*
Solver
-_T_end : TP
-_nts : int
+Solver(T_end : const TS&, nts : const int)
+solve(odes : System <TS,TP,NS,NP>&) : void
<<use>>
Solver_Explicit_Euler
+Solver_Explicit_Euler(T_end : const TS&, nts : const int)
+solve(odes : System <TS,TP,NS,NP>&) : void
Solver_Implicit_Euler
-nlsol : Nonlinear::Solver<TS, TP, NS, NP>&
+Solver_Implicit_Euler(nlsol : Nonlinear::Solver<TS, TP, NS, NP>&, T_end : const TP&, nts : const int)
+solve(odes : System<TS, TP, NS, NP>&)
System_Implicit_Euler
-odes : System<TS, TP, NS, NP>&
-_x_prev : VT<TS>
-_dt : TP
+System_Implicit_Euler(odes : System<TS, TP, NS, NP>&, x_prev : const VT<TS, NS>&, dt : const TP&)
+f() : VT<TS, NS>
+dfdx() : MT<TS, NS, NS>
Diffusion
const int NS=Utils::Dynamic
const int NP=3
System
+System(ns : const int)
+g() : VT<TS, NS>
+dgdx() : MT<TS, NS, NS>
VT
T
N : int
MT
T
M : int
N : int
Nonlinear
Solver
System
Linear
Solver

Outline

Implementation

Library

```
1  doc
2   Doxyfile
3   Makefile
4
5  include
6   ode_solver_explicit_euler.hpp
7   ode_solver.hpp
8   ode_solver_implicit_euler.hpp
9   ode_system.hpp
10
11 Makefile
12
13 src
14       ode_solver.cpp
15       ode_solver_explicit_euler.cpp
16       ode_solver_implicit_euler.cpp
17       ode_system.cpp
```

Implementation

ode_system.hpp


```
1  #include <Eigen/Dense>
2
3  namespace Ode {
4
5  template<typename TS, typename TP, int NS, int NP>
6  class System {
7  public:
8    using VTS=Eigen::Matrix<TS,NS,1>;
9    using MTS=Eigen::Matrix<TS,NS,NS>;
10   using VTP=Eigen::Matrix<TP,NP,1>;
11 protected:
12   VTS _x; VTP _p;
13 public:
14   System(int, int);
15   int ns(); int np();
16   VTS& x(); VTP& p();
17   virtual VTS g()=0; virtual MTS dgdx()=0;
18 };
19
20 }
21
22 #include "../src/ode_system.cpp"
```

Implementation

ode_system.cpp


```
1  namespace Ode {
2
3  template<typename TS, typename TP, int NS, int NP>
4  System<TS,TP,NS,NP>::
5  System(int ns, int np) : _x(ns), _p(np) {}
6
7  template<typename TS, typename TP, int NS, int NP>
8  int System<TS,TP,NS,NP>::ns() { return _x.size(); }
9
10 template<typename TS, typename TP, int NS, int NP>
11 int System<TS,TP,NS,NP>::np() { return _p.size(); }
12
13 template<typename TS, typename TP, int NS, int NP>
14 typename System<TS,TP,NS,NP>::VTS& System<TS,TP,NS,NP>::x() { return _x; }
15
16 template<typename TS, typename TP, int NS, int NP>
17 typename System<TS,TP,NS,NP>::VTP& System<TS,TP,NS,NP>::p() { return _p; }
18
19 }
```

Implementation

ode_solver.hpp


```
1  #include "ode_system.hpp"
2
3  namespace Ode {
4
5  template<typename TS, typename TP, int NS, int NP>
6  class Solver {
7  protected:
8    TP _T_end;
9    int _nts;
10 public:
11   Solver(const TP &, const int);
12   virtual void solve(System<TS,TP,NS,NP>&)=0;
13 };
14
15 }
16
17 #include "../src/ode_solver.cpp"
```

Implementation

ode_solver.hpp


```
1  namespace Ode {
2
3  template<typename TS, typename TP, int NS, int NP>
4  Solver<TS,TP,NS,NP>::Solver(const TP &T_end, const int nts)
5     : _T_end(T_end), _nts(nts) {}
6
7  }
```

Implementation

ode_solver_explicit_euler.hpp


```
1  namespace Ode {
2
3  template<typename TS, typename TP, int NS, int NP>
4  class Solver_Explicit_Euler : public Solver<TS,TP,NS,NP> {
5     using Solver<TS,TP,NS,NP>::_T_end;
6     using Solver<TS,TP,NS,NP>::_nts;
7  public:
8     Solver_Explicit_Euler(const TP &, const int);
9     void solve(System<TS,TP,NS,NP>&);
10 };
11
12 }
13
14 #include "../src/ode_solver_explicit_euler.cpp"
```

Implementation

ode_solver_explicit_euler.cpp


```
1  namespace Ode {
2
3  template<typename TS, typename TP, int NS, int NP>
4  Solver_Explicit_Euler<TS,TP,NS,NP>::
5  Solver_Explicit_Euler(const TP &T_end, const int nts) : Solver<TS,TP,NS,NP>(T_end,nts)
       {}
6
7  template<typename TS, typename TP, int NS, int NP>
8  void
9  Solver_Explicit_Euler<TS,TP,NS,NP>::
10 solve(System<TS,TP,NS,NP>& odes) {
11   for (int i=0;i<_nts;i++) odes.x()=odes.x()+_T_end/_nts*odes.g();
12 }
13
14 }
```

There is nothing to do for a C++ template ("header-only") library.

See code for implementation of implicit Euler method.

Outline

Application

Overview

```
.
 diffusion_explicit_euler.cpp
 diffusion_implicit_euler.cpp

 doc
  Doxyfile
  Makefile

 include
  ode_system_diffusion.hpp
  ode_system_lotka_volterra.hpp

 lotka_volterra_explicit_euler.cpp
 lotka_volterra_implicit_euler.cpp

 Makefile

 src
      ode_system_diffusion.cpp
      ode_system_lotka_volterra.cpp
```

Application

Lotka-Volterra Equation (ode_system_lotka_volterra.hpp)


```
1  #include "ode_system.hpp"
2
3  namespace Lotka_Volterra {
4
5  const int NS=2;
6  const int NP=4;
7
8  template<typename TS, typename TP>
9  class System : public Ode::System<TS,TP,NS,NP> {
10   using Ode::System<TS,TP,NS,NP>::_x;
11   using Ode::System<TS,TP,NS,NP>::_p;
12 public:
13   System();
14   typename Ode::System<TS,TP,NS,NP>::VTS g();
15   typename Ode::System<TS,TP,NS,NP>::MTS dgdx();
16 };
17
18 }
19
20 #include "../src/ode_system_lotka_volterra.cpp"
```

Application

Lotka-Volterra Equation (ode_system_lotka_volterra.cpp)


```
1  namespace Lotka_Volterra {
2
3  template<typename TS, typename TP>
4  System<TS,TP>::System() : Ode::System<TS,TP,NS,NP>(NS,NP) {}
5
6  template<typename TS, typename TP>
7  typename Ode::System<TS,TP,NS,NP>::VTS
8  System<TS,TP>::g() {
9    typename Ode::System<TS,TP,NS,NP>::VTS r;
10   r(0)=(_p(0)-_p(1)*_x(1))*_x(0); r(1)=-(_p(2)-_p(3)*_x(0))*_x(1);
11   return r;
12 }
13
14 template<typename TS, typename TP>
15 typename Ode::System<TS,TP,NS,NP>::MTS
16 System<TS,TP>::dgdx() {
17   typename Ode::System<TS,TP,NS,NP>::MTS drdx;
18   drdx(0,0)=_p(0)-_p(1)*_x(1); drdx(0,1)=-_p(1)*_x(0);
19   drdx(1,0)=_p(3)*_x(1); drdx(1,1)=-(_p(2)-_p(3)*_x(0));
20   return drdx;
21 }
22 }
```

Application

Lotka-Volterra Equation (lotka_volterra_explicit_euler.cpp)


```
1  #include <iostream>
2  #include <cassert>
3
4  #include "ode_system_lotka_volterra.hpp"
5  #include "ode_solver_explicit_euler.hpp"
6
7  int main(int argc, char *argv[]) {
8    assert(argc==3);
9    using T=double;
10   const int NS=Lotka_Volterra::NS, NP=Lotka_Volterra::NP;
11   T T_end=std::stof(argv[1]); assert(T_end>0);
12   int nts=std::stoi(argv[2]); assert(nts>0);
13   Lotka_Volterra::System<T,T> odesys;
14   Ode::Solver_Explicit_Euler<T,T,NS,NP> odesol(T_end,nts);
15   odesys.p()=Ode::System<T,T,NS,NP>::VTP::Random(odesys.np());
16   odesys.x()=Ode::System<T,T,NS,NP>::VTS::Random(odesys.ns());
17   odesol.solve(odesys);
18   std::cout << odesys.x() << std::endl;
19   return 0;
20 }
```

Application

Building

```
1  EXE=$(addsuffix .exe, $(basename $(wildcard *.cpp)))
2  CPPC=g++
3  CPPC_FLAGS=-Wall -Wextra -pedantic -Ofast -march=native
4  EIGEN_DIR=$(HOME)/Software/Eigen
5
6  BASE_DIR=$(HOME)/Documents/gitlab/e-learning/SP_CES/code
7  LIBLS_INC_DIR=$(BASE_DIR)/LINEAR_SYSTEM/libls/include
8  LIBNLS_INC_DIR=$(BASE_DIR)/NONLINEAR_SYSTEM/libnls/include
9  LIBODES_INC_DIR=$(BASE_DIR)/ODE_SYSTEM/libodes/include
10 LIBODES_APPS_INC_DIR=$(BASE_DIR)/ODE_SYSTEM/libodes_apps/include
11
12 all : $(EXE)
13
14 %.exe : %.cpp
15         $(CPPC) $(CPPC_FLAGS) -I$(EIGEN_DIR) -I$(LIBLS_INC_DIR) -I$(
               LIBNLS_INC_DIR) -I$(LIBODES_INC_DIR) -I$(LIBODES_APPS_INC_DIR) $<
               -o $@
16
17 clean :
18         rm -fr $(EXE)
19
20 .PHONY: all clean
```

Outline

Summary

- Discussion of requirements, design and implementation of a type-generic solution infrastructure for parameterized systems of explicit ordinary differential equations

Next Steps

- Download, build and run the sample code.
- Inspect the sample code.
- Continue the course to find out more ...