Software Lab Computational Engineering Science

Type-Generic Solution for Systems of Nonlinear 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 nonlinear 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 parameterized systems of nonlinear equations

$$\mathbf{y} = F(\mathbf{x}, \mathbf{p}) = 0$$

for $\mathbf{x} = \mathbf{x}(\mathbf{p})$ and with invertible Jacobians

$$F' = \frac{dF}{d\mathbf{x}}(\mathbf{x}, \mathbf{p}) \in \boldsymbol{R}^{n_s \times n_s}$$

of the residual $F : \boldsymbol{R}^{n_s} \times \boldsymbol{R}^{n_p} \to \boldsymbol{R}^{n_s}$ at all pairs of states $\mathbf{x}$ and parameters $\mathbf{p}$ of interest.

The Newton method iteratively computes the fixed point

$$\mathbf{x} = \mathbf{x} - \left(F'(\mathbf{x}, \mathbf{p})\right)^{-1} \cdot F(\mathbf{x}, \mathbf{p}) \; .$$

See modules I and II on Newton method.

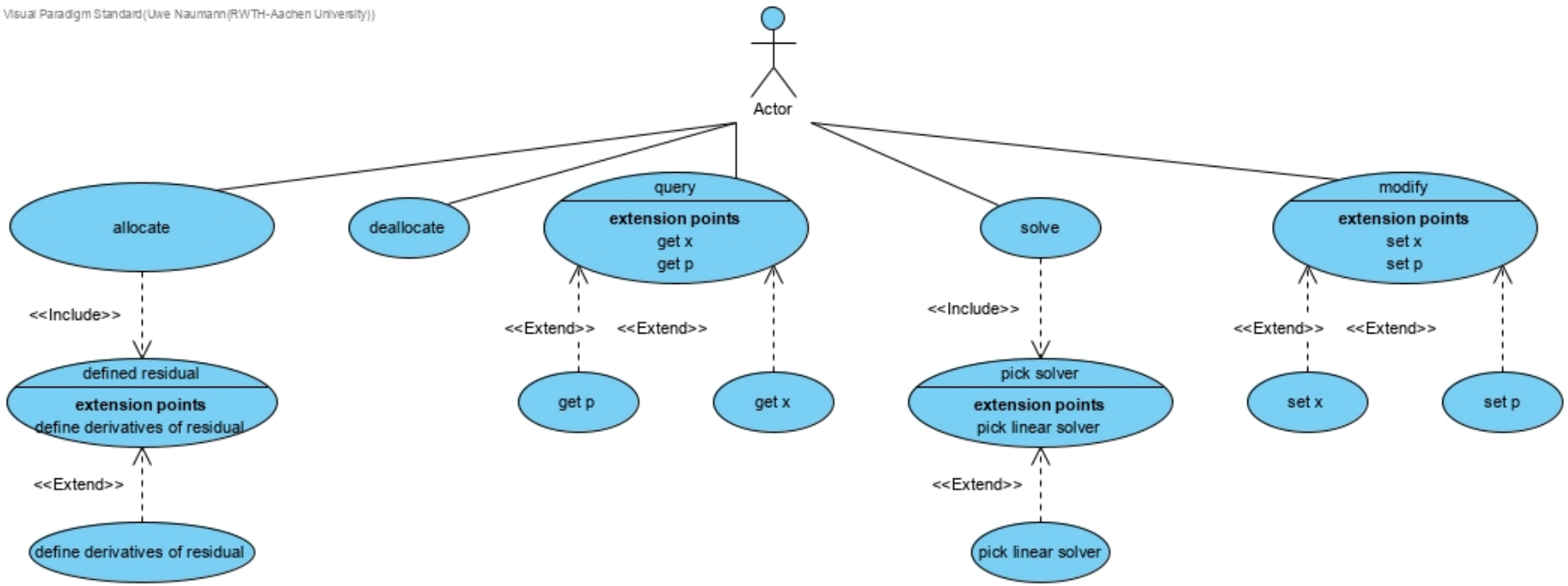
Outline

I am looking for a software library for solving parameterized systems of nonlinear equations $F(\mathbf{x}(\mathbf{p}), \mathbf{p}) = 0$ allowing me to

- define the residual $F : \boldsymbol{R}^{n_s} \times \boldsymbol{R}^{n_p} \rightarrow \boldsymbol{R}^{n_s}$ as well as potentially required derivatives thereof
- define, store and extract the state $\mathbf{x} \in \boldsymbol{R}^{n_s}$ and the parameters $\mathbf{p} \in \boldsymbol{R}^{n_p}$
- extend the choice of nonlinear solvers (in addition to the Newton method)

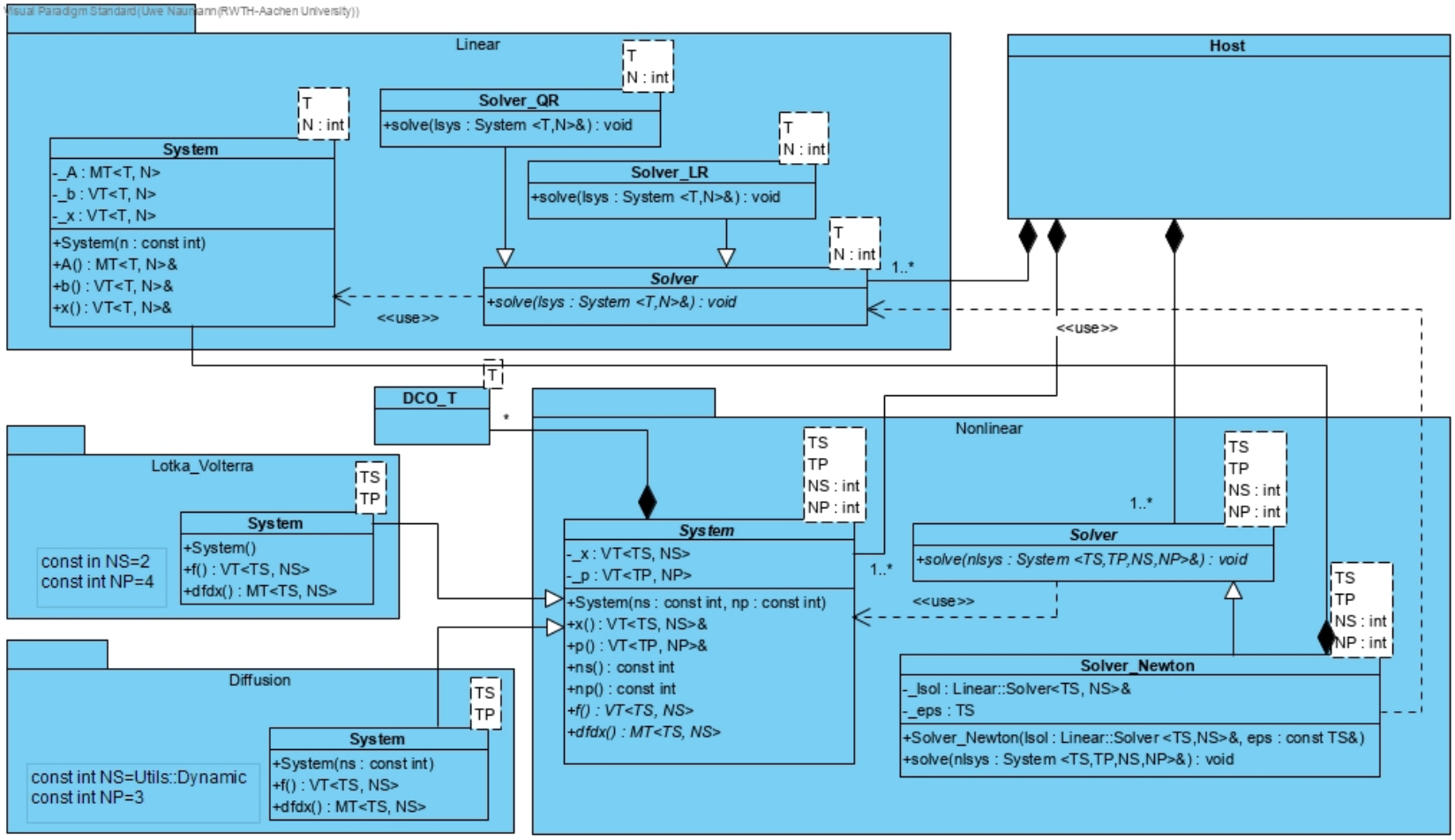
Algorithmic differentiation with dco/c++ shall be used for sensitivity analysis of functions of $\mathbf{x}$ (e.g, $\|\mathbf{x}\|_2$) wrt. to perturbations in $\mathbf{p}$.

Most likely, it will have to be embedded into other software solutions, e.g, for the solution of systems of ordinary differential equations at some later stage.


Visual Paradigm Standard(Uwe Naumann(RWTH-Aachen University))
Actor
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 linear solver
set x
set p
<<Extend>>
<<Extend>>
define derivatives of residual
pick linear solver

- nonlinear system
  - allocation
  - 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 solver
  - allocation
  - deallocation
  - solution of nonlinear system (e.g, Newton using libls)
  - abstraction for extensibility
  - overloading

Outline


Visual Paradigm Standard(Uwe Naumann(RWTH-Aachen University))
Linear
T
N : int
Solver_QR
+solve(lsys : System <T,N>&) : void
Solver_LR
+solve(lsys : System <T,N>&) : void
System
-_A : MT<T, N>
-_b : VT<T, N>
-_x : VT<T, N>
+System(n : const int)
+A() : MT<T, N>&
+b() : VT<T, N>&
+x() : VT<T, N>&
Solver
+solve(lsys : System <T,N>&) : void
1..*
<<use>>
Host
DCO_T
Lotka_Volterra
TS
TP
System
+System()
+f() : VT<TS, NS>
+dfdx() : MT<TS, NS>
const in NS=2
const int NP=4
Nonlinear
NS : int
NP : int
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
+f() : VT<TS, NS>
+dfdx() : MT<TS, NS>
Solver
+solve(nlsys : System <TS,TP,NS,NP>&) : void
Solver_Newton
-_lsol : Linear::Solver<TS, NS>&
-_eps : TS
+Solver_Newton(lsol : Linear::Solver <TS,NS>&, eps : const TS&)
+solve(nlsys : System <TS,TP,NS,NP>&) : void
Diffusion
System
+System(ns : const int)
+f() : VT<TS, NS>
+dfdx() : MT<TS, NS>
const int NS=Utils::Dynamic
const int NP=3

Outline

Implementation

Library

```
doc
 Doxyfile
 Makefile

include
 nonlinear_solver.hpp
 nonlinear_solver_newton.hpp
 nonlinear_system.hpp

Makefile

src
      nonlinear_solver_newton.cpp
      nonlinear_system.cpp
```

Implementation

nonlinear_system.hpp


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

Implementation

nonlinear_system.cpp


```
1  namespace Nonlinear {
2
3  template<typename TS, typename TP, int NS, int NP>
4  System<TS,TP,NS,NP>::System(int ns, int np)
5    : _x(VTS::Zero(ns))
6    , _p(VTP::Zero(np)) {}
7
8  template<typename TS, typename TP, int NS, int NP>
9  int System<TS,TP,NS,NP>::ns() { return _x.size(); }
10
11 template<typename TS, typename TP, int NS, int NP>
12 int System<TS,TP,NS,NP>::np() { return _p.size(); }
13
14 template<typename TS, typename TP, int NS, int NP>
15 typename System<TS,TP,NS,NP>::VTS&
16 System<TS,TP,NS,NP>::x() { return _x; }
17
18 template<typename TS, typename TP, int NS, int NP>
19 typename System<TS,TP,NS,NP>::VTP&
20 System<TS,TP,NS,NP>::p() { return _p; }
21
22 }
```

Implementation

nonlinear_solver.hpp


```
1  #include "nonlinear_system.hpp"
2
3  namespace Nonlinear {
4
5  template<typename TS, typename TP, int NS, int NP>
6  class Solver {
7  public:
8    virtual void solve(System<TS,TP,NS,NP>&)=0;
9  };
10
11 }
```

Implementation

nonlinear_solver_newton.hpp


```
1  #include "linear_solver.hpp"
2  #include "nonlinear_system.hpp"
3  #include "nonlinear_solver.hpp"
4
5  namespace Nonlinear {
6
7  template<typename TS, typename TP, int NS, int NP>
8  class Solver_Newton : public Solver<TS,TP,NS,NP> {
9    Linear::Solver<TS,NS>& _lsol;
10   TS _eps;
11 public:
12   Solver_Newton(Linear::Solver<TS,NS>&, const TS&);
13   void solve(System<TS,TP,NS,NP>&);
14 };
15
16 }
17
18 #include "../src/nonlinear_solver_newton.cpp"
```

Implementation

nonlinear_solver_newton.cpp


```
1  namespace Nonlinear {
2
3  template<typename TS, typename TP, int NS, int NP>
4  Solver_Newton<TS,TP,NS,NP>::Solver_Newton(Linear::Solver<TS,NS>& lsol, const TS&
       eps) : _lsol(lsol), _eps(eps) {}
5
6  template<typename TS, typename TP, int NS, int NP>
7  void
8  Solver_Newton<TS,TP,NS,NP>::solve(System<TS,TP,NS,NP>& nlsys) {
9    Linear::System<TS,NS> lsys(nlsys.ns());
10   while (nlsys.f().norm()>_eps) {
11     lsys.A()=nlsys.dfdx(); lsys.b()=-nlsys.f();
12     _lsol.solve(lsys);
13     nlsys.x()+=lsys.x();
14   }
15 }
16
17 }
```

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

Outline

Application

Overview

```
doc
 Doxyfile
 Makefile

include
 nonlinear_system_diffusion.hpp
 nonlinear_system_lotka_volterra.hpp
 nonlinear_system_toy.hpp

Makefile

src
 nonlinear_system_diffusion.cpp
 nonlinear_system_lotka_volterra.cpp
 nonlinear_system_toy.cpp

toy.cpp
diffusion.cpp
lotka_volterra.cpp
```

Application

Toy Problem: $x^2 - p = 0$ (nonlinear_system_toy.hpp)


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

Application

Toy Problem: $x^2 - p = 0$ (nonlinear_system_toy.cpp)


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

Application

Toy Problem: $x^2 - p = 0$ (toy.cpp)


```
1  #include<iostream>
2
3  #include "nonlinear_system_toy.hpp"
4  #include "linear_solver_qr.hpp"
5  #include "nonlinear_solver_newton.hpp"
6
7  int main() {
8    using T=double;
9    Toy::System<T,T> nlsys;
10   Linear::Solver_QR<T,Toy::NS> lsol;
11   Nonlinear::Solver_Newton<T,T,Toy::NS,Toy::NP> nlsol(lsol,1e-7);
12   nlsys.x()(0)=1;
13   nlsys.p()(0)=2;
14   nlsol.solve(nlsys);
15   std::cout << "x=" << nlsys.x() << std::endl;
16   return 0;
17 }
```

```
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=$(BASE_DIR)/NONLINEAR_SYSTEM/libnls/include
9  LIBNLS_APPS_INC_DIR=$(BASE_DIR)/NONLINEAR_SYSTEM/libnls_apps/include
10
11 all : $(EXE)
12
13 %.exe : %.cpp
14         $(CPPC) $(CPPC_FLAGS) -I$(EIGEN_DIR) -I$(LIBLS_INC_DIR) -I$(
               LIBNLS_INC_DIR) -I$(LIBNLS_APPS_INC_DIR) $< -o $@
15
16 clean :
17         rm -fr $(EXE)
18
19 .PHONY: all clean
```

Outline

We consider the algorithmic differentiation of the solution of $x^2 - p = 0$ wrt. p in with dco/c++.

Setting $p^{(1)} = 1$ in

$$x^{(1)} = \frac{dx}{dp} \cdot p^{(1)}$$

yields the derivative as the tangent of the state x.

See module I on algorithmic differentiation and source code.

```
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  DCO_DIR=$(HOME)/Software/dco
6  DCO_INC_DIR=$(DCO_DIR)/include
7  DCO_LIB_DIR=$(DCO_DIR)/lib
8  DCO_FLAGS=-DDCO_DISABLE_AUTO_WARNING
9  DCO_LIB=dcoc
10
11 BASE_DIR=$(HOME)/Documents/gitlab/e-learning/SP_CES/code
12 LIBLS_INC_DIR=$(BASE_DIR)/LINEAR_SYSTEM/libls/include
13 LIBNLS_INC_DIR=$(BASE_DIR)/NONLINEAR_SYSTEM/libnls/include
14 LIBNLS_APPS_INC_DIR=$(BASE_DIR)/NONLINEAR_SYSTEM/libnls_apps/include
15
16 all : $(EXE)
17
18 %.exe : %.cpp
19         $(CPPC) $(CPPC_FLAGS) $(DCO_FLAGS) -I$(EIGEN_DIR) -I$(DCO_INC_DIR)
             -I$(LIBLS_INC_DIR) -I$(LIBNLS_INC_DIR) -I$(LIBNLS_APPS_INC_DIR) -L$(
             DCO_LIB_DIR) $< -o $@ -l$(DCO_LIB)
20 ...
```

Outline

Summary

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

Next Steps

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