Software Lab Computational Engineering Science

Type-Generic Solution for Systems of Linear 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 systems of linear 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

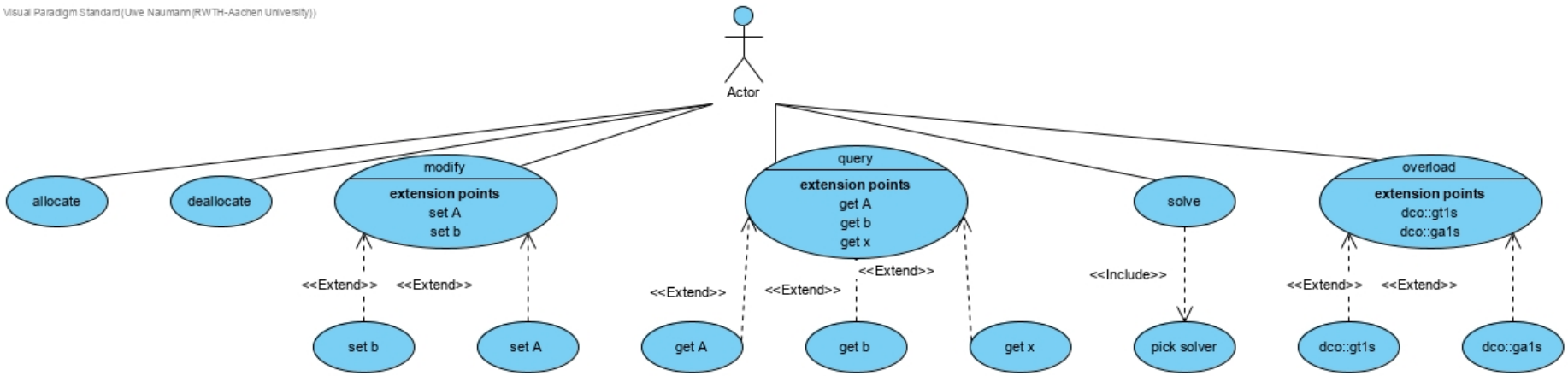
I am looking for a software library for solving systems of linear equations $A \cdot \mathbf{x} = \mathbf{b}$ including

- definition, storage and extraction of $A \in \boldsymbol{R}^{n \times n}$ and $\mathbf{b} \in \boldsymbol{R}^{n}$
- demonstrated extensible choice of direct linear solvers
- storage and extraction of solution $\mathbf{x}$.

The software should run efficiently on the RWTH Compute Cluster.

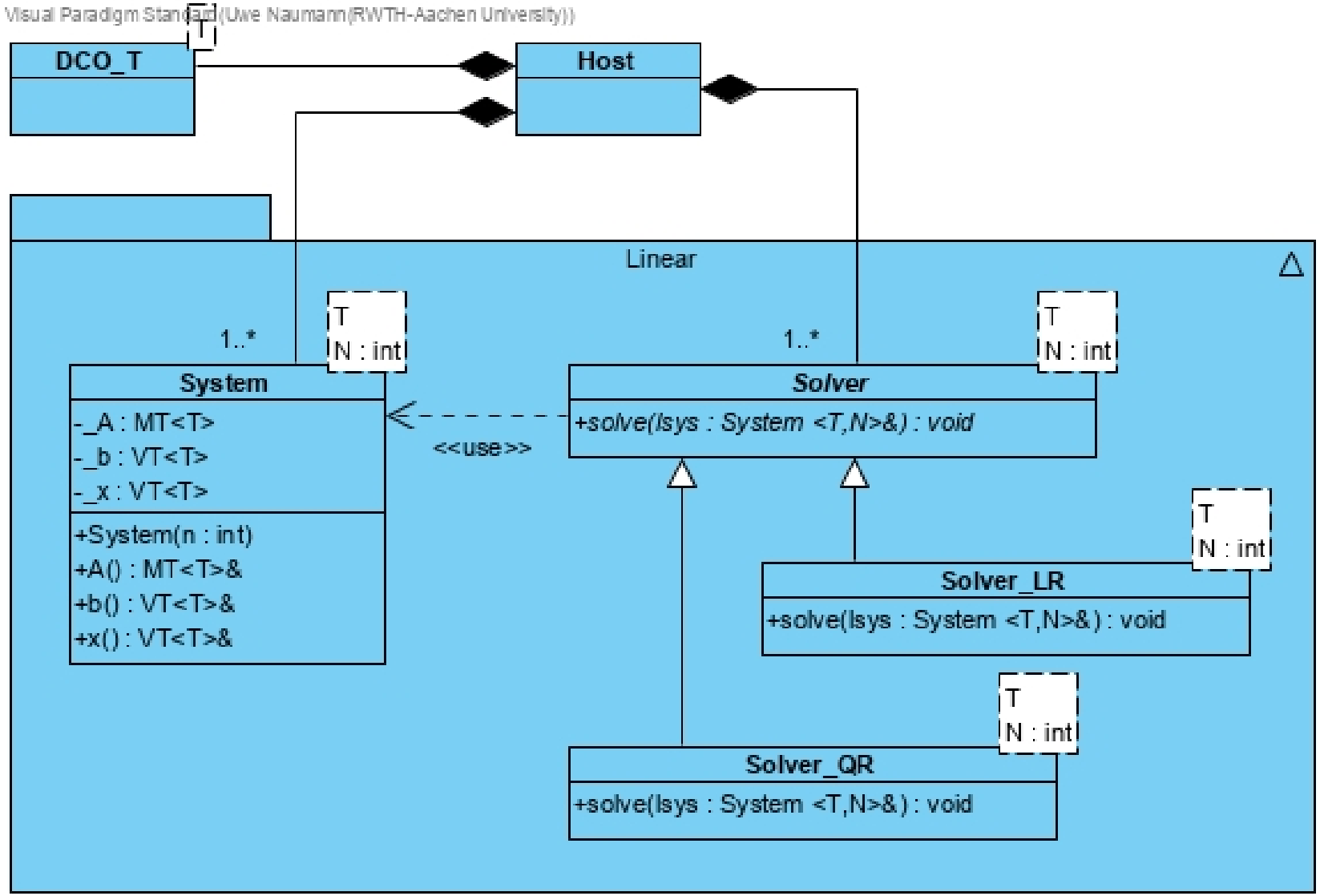
Algorithmic differentiation with dco/c++ shall be used to estimate the condition of the system as well as for sensitivity analysis of functions of $\mathbf{x}$ (e.g, $\|\mathbf{x}\|_2$) wrt. to perturbations of A and/or $\mathbf{b}$.

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


Visual Paradigm Standard(Uwe Naumann(RWTH-Aachen University))
Actor
allocate
deallocate
modify
extension points
set A
set b
query
extension points
get A
get b
get x
solve
overload
extension points
dco::gt1s
dco::ga1s
<<Extend>>
<<Extend>>
<<Extend>>
<<Extend>>
<<Extend>>
<<Include>>
<<Extend>>
<<Extend>>
set b
set A
get A
get b
get x
pick solver
dco::gt1s
dco::ga1s

- linear system
  - allocation
  - deallocation
  - generic element type (T) of elements of $A, \mathbf{b}, \mathbf{x}$ enabling overloading
  - matrix type (MT) for storage of A
  - vector type (VT) for storage of $\mathbf{b}$ and $\mathbf{x}$
  - read/write access routines for $A, \mathbf{b}, \mathbf{x}$
- linear solver
  - allocation
  - deallocation
  - solution of linear system
  - abstraction for extensibility
  - implementation of two direct solvers (e.g, LR and QR factorization)
  - overloading

Outline


Visual Paradigm Standard (Uwe Naumann(RWTH-Aachen University))
T
DCO_T
Host
Linear
1..*
T
N : int
System
-_A : MT<T>
-_b : VT<T>
-_x : VT<T>
+System(n : int)
+A() : MT<T>&
+b() : VT<T>&
+x() : VT<T>&
<<use>>
1..*
T
N : int
Solver
+solve(lsys : System <T,N>&) : void
T
N : int
Solver_LR
+solve(lsys : System <T,N>&) : void
T
N : int
Solver_QR
+solve(lsys : System <T,N>&) : void

Outline

Implementation

Library

```
1  doc // doxygen documentation
2   Doxyfile
3   Makefile
4
5  include // type-generic versions of
6   linear_solver.hpp // abstract linear solver
7   linear_solver_lr.hpp // LR factorization
8   linear_solver_qr.hpp // QR factorization
9   linear_system.hpp // linear system
10  utils.hpp // utilities
11
12 Makefile // top-level build script
13
14 src // template implementations
15  linear_solver_lu.cpp
16  linear_solver_qr.cpp
17  linear_system.cpp
18
19 UML // UML models using Visual Paradigm
```

Implementation

linear_system.hpp


```
1  #include <Eigen/Dense>
2
3  namespace Linear {
4
5  template<typename T, int N>
6  class System {
7  public:
8    using MT=Eigen::Matrix<T,N,N>;
9    using VT=Eigen::Matrix<T,N,1>;
10
11 protected:
12   VT _x,_b; MT _A;
13
14 public:
15   System(int);
16   VT& x(); VT& b(); MT& A();
17 };
18
19 }
20
21 #include "../src/linear_system.cpp"
```

Implementation

linear_system.cpp


```
1  namespace Linear {
2
3  template<typename T, int N>
4  System<T,N>::System(int n)
5      : _x(VT::Zero(n)), _b(VT::Zero(n)), _A(MT::Zero(n,n)) {}
6
7  template<typename T, int N>
8  typename System<T,N>::VT&
9  System<T,N>::x() { return _x; }
10
11 template<typename T, int N>
12 typename System<T,N>::VT&
13 System<T,N>::b() { return _b; }
14
15 template<typename T, int N>
16 typename System<T,N>::MT&
17 System<T,N>::A() { return _A; }
18
19 }
```

Implementation

linear_solver.hpp


```
1  #include "linear_system.hpp"
2
3  namespace Linear {
4
5  template<typename T, int N>
6  struct Solver {
7     virtual void solve(System<T,N>&)=0;
8  };
9
10 }
```

Implementation

linear_solver_qr.hpp


```
1  #include "linear_system.hpp"
2  #include "linear_solver.hpp"
3
4  namespace Linear {
5
6  template<typename T, int N>
7  class Solver_QR : public Solver<T,N> {
8
9  public:
10   void solve(System<T,N>&);
11 };
12
13 }
14
15 #include "../src/linear_solver_qr.cpp"
```

Implementation

linear_solver_qr.cpp


```
1  namespace Linear {
2
3  template<typename T, int N>
4  void Solver_QR<T,N>::solve(System<T,N>& s) {
5     Eigen::HouseholderQR<typename System<T,N>::MT> QR(s.A());
6     s.x()=QR.solve(s.b());
7  }
8
9  }
```

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

Outline

Application

Overview


```
doc // doxygen documentation
 Doxyfile
 Makefile

linear_system_lr.cpp // test: LR factorization

linear_system_qr.cpp // test: QR factorization

linear_system_tangent.cpp // test: overloading with dco::gt1s<T>::type

linear_system_adjoint.cpp // test: overloading with dco::ga1s<T>::type

linear_system_condition.cpp // test: estimation of system condition using dco/c++

Makefile // top-level build script
```

```
1  #include "linear_system.hpp"
2  #include "linear_solver_qr.hpp"
3  #include "utils.hpp"
4
5  #include<cassert>
6  #include<iostream>
7
8  int main(int argc, char* argv[]) {
9    assert(argc==2); int n=std::stoi(argv[1]);
10   using T=double;
11   Linear::System<T,Utils::Dynamic> lsys(n); // allocation
12   lsys.A()=Linear::System<T,Utils::Dynamic>::MT::Random(n,n); // write
13   lsys.b()=Linear::System<T,Utils::Dynamic>::VT::Random(n); // ... access
14   Linear::Solver_QR<T,Utils::Dynamic> lsol; // allocation
15   lsol.solve(lsys); // solve linear system
16   std::cout << "x=" << lsys.x() << std::endl; // read access
17   return 0; // deallocation (automatically)
18 }
```

```
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  LIBLS_DIR=$(PWD)/../libls
6  LIBLS_INC_DIR=$(LIBLS_DIR)/include
7
8  all : $(EXE)
9
10 %.exe : %.cpp
11         $(CPPC) $(CPPC_FLAGS) -I$(EIGEN_DIR) -I$(LIBLS_INC_DIR) $< -o $@
12
13 clean :
14         rm -fr $(EXE)
15
16 .PHONY: all clean
```

Consider the type-generic class

```
1 template<typename T, int N>
2 class Linear_System { ... }
```

Dynamic instantiation yields need for dynamic memory management (use of heap), e.g,

```
1 int main(int argc, char* argv[]) {
2   assert(argc==2); int n=std::stoi(argv[1]);
3   Linear::System<double,Utils::Dynamic> lsys(n); ... }
```

Static instantiation enables static memory management (use of stack), e.g,

```
1   const int N=3;
2   Linear::System<double,N> lsys(N); ... }
```

Statically instantiated code runs typically faster than dynamically instantiated code.

Outline

The (relative) condition of $A \cdot \mathbf{x} = \mathbf{b}$ is evaluated as

$$\text{cond}(A) = \|A\|_2 \cdot \|A^{-1}\|_2 \quad .$$

From

$$\mathbf{x} = A^{-1} \cdot \mathbf{b} \quad \Rightarrow \quad \frac{d\mathbf{x}}{d\mathbf{b}} = A^{-1}$$

follows a (suboptimal) method for computing cond(A) using algorithmic differentiation of the linear solver wrt. $\mathbf{b}$.

The additional functional requirement

- L_2-norm of objects of type MT

is provided by Eigen.

See source code.

We consider the algorithmic differentiation of

$$y = f(A, \mathbf{b}) = \|\mathbf{x}\| = \|A^{-1} \cdot \mathbf{b}\| : \boldsymbol{R}^{n \times n} \times \boldsymbol{R}^n \to \boldsymbol{R}$$

wrt. $\mathbf{b}$ in adjoint mode with dco/c++.

Setting $y_{(1)} = 1$ in

$$\mathbf{b}_{(1)} = y_{(1)} \cdot \frac{df}{d\mathbf{b}}$$

yields the gradient as the adjoint of $\mathbf{b}$ efficiently ($O(1) \cdot \text{Cost}(f)$). Both tangent mode and finte differences induce computational costs of $O(n) \cdot \text{Cost}(f)$.

See module III 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=$(DCO_DIR)/lib
8  DCO_FLAGS=-DDCO_DISABLE_AUTO_WARNING
9  DCO_LIB=dcoc
10 LIBLS_DIR=$(PWD)/../libls
11 LIBLS_INC_DIR=$(LIBLS_DIR)/include
12
13 all : $(EXE)
14
15 %.exe : %.cpp
16         $(CPPC) $(CPPC_FLAGS) $(DCO_FLAGS) -I$(EIGEN_DIR) -I$(DCO_INC_DIR)
               -I$(LIBLS_INC_DIR) -L$(DCO_LIB_DIR) $< -o $@ -l$(DCO_LIB)
17
18 clean :
19         rm -fr $(EXE)
20
21 .PHONY: all clean
```

Outline

Summary

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

Next Steps

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