Software Lab Computational Engineering Science

Overview of Sample Code and Basic Solution for Systems of Linear Equations

Uwe Naumann

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

RWTH Aachen University

Contents

Outline

Objective

- Overview of the sample code used as a basis for the tutorial exercises; introduction to design and implementation of a basic solution infrastructure for systems of linear equations

Learning Outcomes

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

Outline

We discuss the design of a software for the solution of

- systems of linear equations
  - basic implementation enables
    - sensitivity analysis by finite differences
    - estimation of condition
  - type-generic implementation enables
    - sensitivity analysis by tangent and adjoint modes of dco/c++
    - estimation of condition
- systems of nonlinear equations using linear solver
- systems of explicit ordinary differential equations using nonlinear solver.

Tutorial exercises require modification of the given sample software.

- use of C++ as programming language
- development and execution under Linux (RWTH Compute Cluster[1])
- compilation using g++
- build system using make[2]
- algorithmic differentiation with dco/c++[3]
- source code documentation with doxygen[4]

[1] https://doc.itc.rwth-aachen.de/display/CC/Home
[2] https://www.gnu.org/software/make/
[3] www.nag.co.uk/content/algorithmic-differentiation-software
[4] http://www.doxygen.nl

Outline

Find $\mathbf{x} \in \boldsymbol{R}^n$ such that $A \cdot \mathbf{x} = \mathbf{b} \in \boldsymbol{R}^n$ implying $\mathbf{x} = A^{-1} \cdot \mathbf{b}$ and requiring $A \in \boldsymbol{R}^{n \times n}$ to be invertible. Direct methods include

- *LR* factorization

$$A \cdot \mathbf{x} = (L \cdot R) \cdot \mathbf{x} = L \cdot (R \cdot \mathbf{x}) = \mathbf{b} \Rightarrow R \cdot \mathbf{x} = L^{-1} \cdot \mathbf{b}$$

  with lower unitriangular $L \in \boldsymbol{R}^{n \times n}$ (forward substitution) and upper triangular $R \in \boldsymbol{R}^{n \times n}$ (backward substitution)

- *QR* factorization

$$A \cdot \mathbf{x} = (Q \cdot R) \cdot \mathbf{x} = Q \cdot (R \cdot \mathbf{x}) = \mathbf{b} \Rightarrow R \cdot \mathbf{x} = Q^{-1} \cdot \mathbf{b} = Q^T \cdot \mathbf{b}$$

  with orthogonal $Q \in \boldsymbol{R}^{n \times n}$ and upper triangular $R \in \boldsymbol{R}^{n \times n}$.

See modules I and II on Linear Algebra

We use Eigen[5] for linear algebra.

[5] eigen.tuxfamily.org

Outline

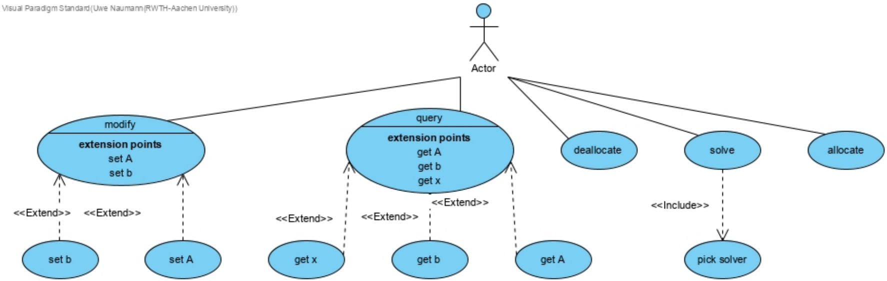
I am looking for a software library for solving systems of linear equations $A \cdot \mathbf{x} = \mathbf{b}$ including

- 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.

Finite difference approximation of first (and higher) derivatives of $\mathbf{x}$ wrt. A and/or $\mathbf{b}$ 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
modify
extension points
set A
set b
query
extension points
get A
get b
get x
deallocate
solve
allocate
<<Extend>>
<<Extend>>
<<Extend>>
<<Extend>>
<<Extend>>
<<Include>>
set b
set A
get x
get b
get A
pick solver

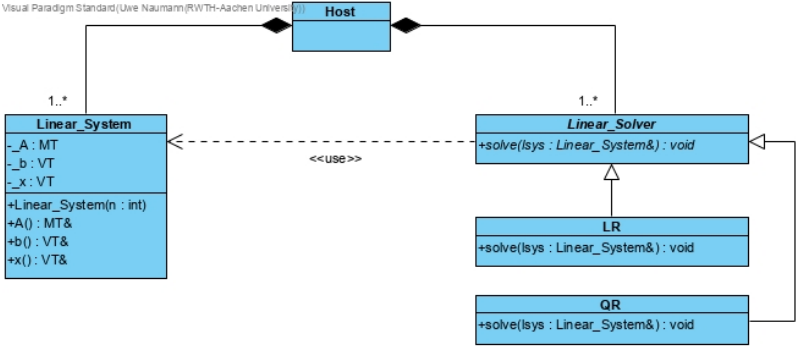
Functional System Requirements

- linear system
  - allocation
  - deallocation
  - fixed element type (T) of elements of $A, \mathbf{b}, \mathbf{x}$
  - 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)

Outline

Design

Class Model



Outline

Implementation

Library

```
doc // doxygen documentation

include
 linear_solver.hpp // abstract linear solver
 linear_solver_lr.hpp // LR factorization
 linear_solver_qr.hpp // QR factorization
 linear_system.hpp // linear system

lib // libls.a ends up here

Makefile // top-level build script

src // implementations
 linear_solver_lr.cpp
 linear_solver_qr.cpp
 linear_system.cpp
 Makefile

UML // UML models using Visual Paradigm
```

Implementation

linear_system.hpp

```
#include <Eigen/Dense>

class Linear_System {

public:
  using T=double;
  using MT=Eigen::Matrix<T,Eigen::Dynamic,Eigen::Dynamic>;
  using VT=Eigen::Matrix<T,Eigen::Dynamic,1>;

protected:
  VT _x, _b; MT _A;

public:
  Linear_System(int);
  VT& x(); VT& b(); MT& A();
};
```

linear_solver[_lr].hpp

- linear_solver.hpp

```
#include "linear_system.hpp"

struct Linear_Solver {
  virtual void solve(Linear_System&)=0;
};
```

- linear_solver_lr.hpp

```
#include "linear_system.hpp"
#include "linear_solver.hpp"

class Linear_Solver_LR : public Linear_Solver {
public:
  void solve(Linear_System&);
};
```

Implementation

Building

```makefile
OBJ=$(addsuffix .o, $(basename $(wildcard *.cpp)))
CPPC=g++
AR=ar -r
CPPC_FLAGS=-Wall -Wextra -pedantic -Ofast -march=native
INC_DIR=../include
EIGEN_DIR=$(HOME)/Software/Eigen

libls.a : $(OBJ)
        $(AR) $@ $^
        mv $@ ../lib

%.o : %.cpp
        $(CPPC) -c $(CPPC_FLAGS) -I$(INC_DIR) -I$(EIGEN_DIR) $< -o $@

clean :
        rm -fr $(OBJ)

.PHONY: clean
```

Outline

Application

Overview

```
doc // doxygen documentation
 Doxyfile
 Makefile

linear_system_condition.cpp // test: estimation of system condition

linear_system_lr.cpp // test: LR factorization

linear_system_qr.cpp // test: QR factorization

Makefile // top-level build script
```

```
#include "linear_system.hpp"
#include "linear_solver_qr.hpp"

#include<cassert>
#include<iostream>

int main(int argc, char* argv[]) {
  assert(argc==2); int n=std::stoi(argv[1]);
  Linear_System lsys(n); // allocation
  lsys.A()=Linear_System::MT::Random(n,n); // write access and ...
  lsys.b()=Linear_System::VT::Random(n); // ... random initialization
  Linear_Solver_QR lsol; // allocation
  lsol.solve(lsys); // solve linear system
  std::cout << "x=" << lsys.x() << std::endl; // read access
  return 0; // deallocation (automatically)
}
```

Application

Building

```
EXE=$(addsuffix .exe, $(basename $(wildcard *.cpp)))
CPPC=g++
CPPC_FLAGS=-Wall -Wextra -pedantic -Ofast -march=native
EIGEN_DIR=$(HOME)/Software/Eigen
LIBLS_DIR=$(PWD)/../libls
LIBLS_INC_DIR=$(LIBLS_DIR)/include
LIBLS_LIB_DIR=$(LIBLS_DIR)/lib
LIBLS=ls

all : $(EXE)

%.exe : %.cpp
	$(CPPC) $(CPPC_FLAGS) -I$(EIGEN_DIR) -I$(LIBLS_INC_DIR) -L$(LIBLS_LIB_DIR) $< -o $@ -l$(LIBLS)

clean :
	rm -fr $(EXE)

.PHONY: all clean
```

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 finite difference approximation of A^1.

The additional functional requirement

- L_2-norm of objects of type MT

is provided by Eigen.

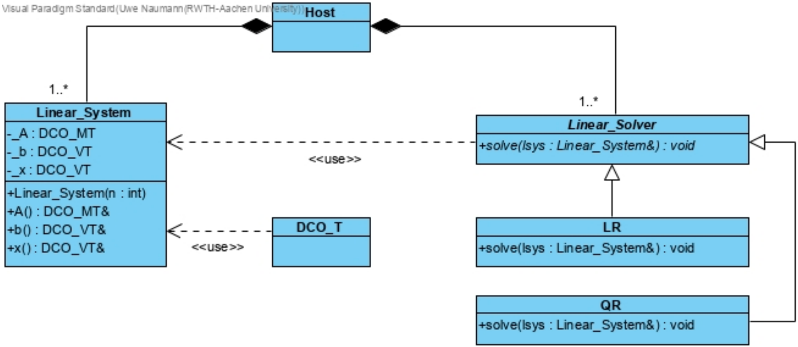
See source code.

Outline

I have heard of this cool technique for computing derivatives of arbitrary differentiable computer programs with machine accuracy (as opposed to finite differences, where finding a suitable magnitude of the perturbation can be "painful." They call it algorithmic differentiation.

One way to implement it is by function and operator overloading for custom data types in C++. People keep telling me about the world's best AD software dco/c++ (:-)). I would like to be able to use it for the computation of A^{-1} in the above case study as well as for other applications requiring first and potentially higher derivatives of $\mathbf{x}$ or of functions of $\mathbf{x}$.

See modules I, II and III on Algorithmic Differentiation.



Duplication of source code to be avoided!

Outline

Summary

- Overview of the sample code used as a basis for the tutorial exercises
- Discussion of requirements, design and implementation of a basic solution infrastructure for systems of linear equations
- Discussion of limitations.

Next Steps

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