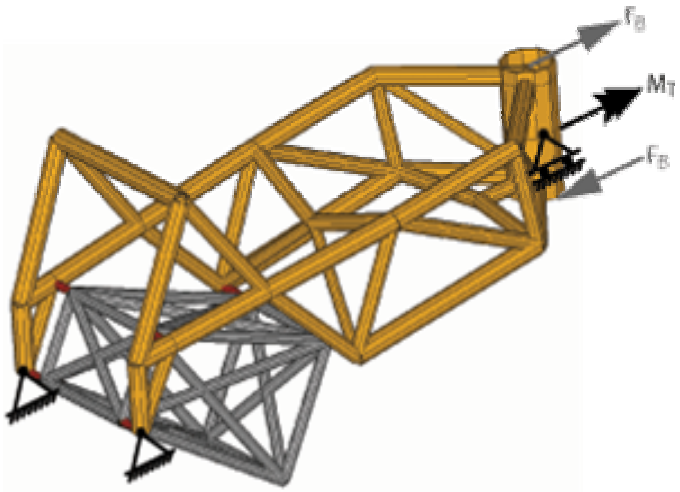
FELyX
Finite Element Library eXperiment

Motorbike Frame

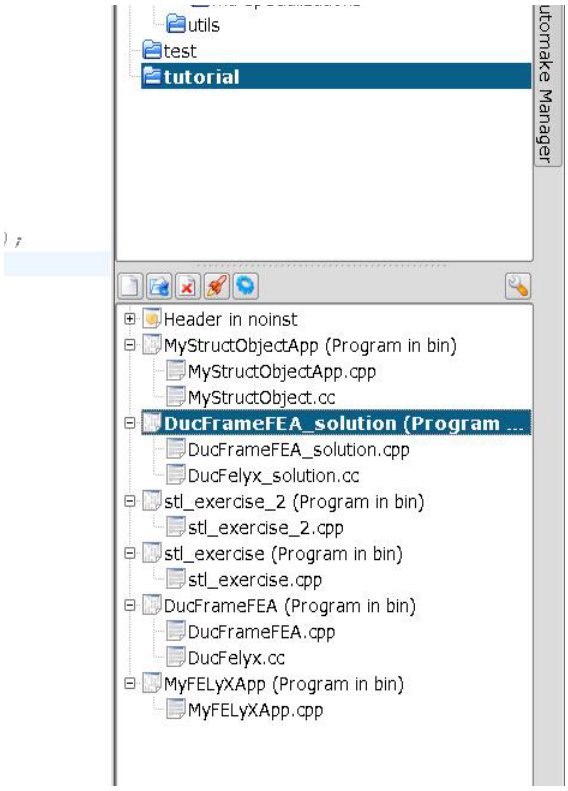
New Object
Calculate Mass
Insert Load Case
Postprocessing



This example is based on “Winschhebel”

Import Model

- The FE Model is generated in ANSYS and exported as FE database (see example Winschhebel). The model includes
  - The whole geometry (Nodes, elements, beam cross sections)
  - Material
- Missing: Forces and boundary conditions
- The aim is to calculate the mass, introduce two loadcases, get max. von Mises stress and torsional stiffness of the frame
- Here we create three files in the folder /felyx/tutorial
  - DucFelyx_solution.h – defines a new object
  - DucFelyx_solution.cc – includes the new functions
  - DucFrameFEA_solution.cpp – application file
- Also a new target “DucFrameFEA_solution” is built. Add DucFelyx_solution.cc and DucFrameFEA_solution.cpp to this target and make it active
- The contents of the new files is described on the next sides



DucFelyx_solution.h

This class defines a new object derived from StructObject. All additional needed functions for this calculations are defined in this header file.

```
//---------------------------------------------------------------------------
// DucFelyx.h
//---------------------------------------------------------------------------

#ifndef DucFelyx_h
#define DucFelyx_h DucFelyx_h

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <string>

#include <cmath>

#include "StructObject.h"

using namespace std;
using namespace felyx;

extern const double PI;

class DucFelyxObject : public StructObject {

  public:

    // Constructors
    // ------------
    DucFelyxObject() : StructObject() {}

    DucFelyxObject( std::string filenam_, std::string datadir_,
unsigned noise_ = 0 )
     : StructObject( filenam_, datadir_, noise_){}

    // Functions to apply different loadcases
    // --------------------------------------
   void ApplyTorsionLoadcase();

   void ApplyBrakingLoadcase();

    // Eval functions
    // --------------

    double EvalMass() const;

    double EvalTorsionStiffness() const;

    double EvalMaxStress();

  private:
};


#endif
```

Include all header files

Header for standard mathematical functions

include FELyX object

Define namespaces;

Definition of class DucFelyxObject
Class is derived from StructObject, implementing all necessary FEM functionality for the ducati frame optimization.

public:

//Constructors
// ------------
Empty constructor, doing nothing

File constructor, reading in FE model
Args: filenam, datadir, noiselevel (optional)

// Functions to apply different loadcases
// --------------------------------------
Function that deletes all forces and moments and applies the torsion loadcase
Function that deletes all forces and moments and applies the braking loadcase
//Eval functions
// --------------
Eval mass of structure in a range of element reference numbers in kg
Run FE-analysis for torsion loadcase
Eval torsion stiffness of frame, including engine in Nm/degree
Eval maximum stress of structure, looking at elements in range of reference numbers

Section of DucFelyx_solution.cc

The new defined functions are implemented here. In this manual the functions to insert a loadcase, eval mass and get max. Van Mises stress are described.

```
///-------------------------------------------------------------------------
// DucFelyx.cc
//-------------------------------------------------------------------------
```

Include all necessary libraries

```
#include "boost/spirit.hpp"
#define RULE(name, definition) typeof(definition) name = definition
namespace bs = boost::spirit;
```

Define namespaces

```
#include "DucFelyx_solution.h"
#include "IOSpiritUtils.h"
#include <stdexcept>
```

Include header files

```
DucFelyxObject::DucFelyxObject( const unsigned noise_, const
unsigned nVarTubes_, const fs::path& path_ )
    : StructObject( noise_ ), nVarTubes( nVarTubes_ ) {

  if ( !fs::exists( path_ ) ) {
    std::string error = "file path does not exist: " +
path_.native_file_string();
    FELYX_RUNTIME_THROW( error.c_str() );
  }

  std::string branch_path = fs::complete( path_
).branch_path().native_file_string() ;
  std::string file = path_.leaf();
  LoadAnsysModel( file, branch_path );
}
```

Constructor that initializes FE model

Delete all forces and moments and apply torsion loadcase

```
void DucFelyxObject::ApplyTorsionLoadcase() {
```

Delete all consisting forces and moments

```
  for ( vector<bc_type>::iterator it = BoundaryConditions.begin();
        it != BoundaryConditions.end(); ++it )
    it->deleteForces();
```

Apply torsional moment

```
  vector<node_type>::iterator nit = find ( Nodes.begin(),
Nodes.end(), node_type( 785.0 , 0.0 , 135.0 ) );
  if ( nit->ExistBoundCon() )
    nit->BoundConPtr->set
    ( Mx, 700000 );
  else
    throw std::logic_error( "ERROR in
DucFelyxObject::ApplyTorsionLoadcase : BoundConPtr not set!" );

  if ( noise > 0 )
    OutStream << "--< " << my_timer.elapsed()
    << " >-- " << " Applied torsion loadcase to node : " << distance(
Nodes.begin(), nit ) << endl;

}
```

End of function “ApplyTorsionalLoadcase()”

// see next page

```
void DucFelyxObject::ApplyBrakingLoadcase() {

  for ( vector<bc_type>::iterator it = BoundaryConditions.begin();
        it != BoundaryConditions.end(); ++it )
    it->deleteForces();

  double Force = 4256.0e3 / 182.5;
  vector<node_type>::iterator nit = find ( Nodes.begin(), Nodes.end(),
node_type( 785.0 , 0.0 , 44.0 ) );
  if ( nit->ExistBoundCon() )
    nit->BoundConPtr->set
    ( Fx, -Force );
  else
    throw std::logic_error( "ERROR in
DucFelyxObject::ApplyBrakingLoadcase : BoundConPtr not set!" );

  if ( noise > 0 )
    OutStream << "--< " << my_timer.elapsed()
    << " >-- " << " Applied braking loadcase to node : " << distance(
Nodes.begin(), nit ) << endl;


  nit = find ( Nodes.begin(), Nodes.end(), node_type( 785.0 , 0.0 , 226.5 )
);
  if ( nit->ExistBoundCon() )
    nit->BoundConPtr->set
    ( Fx, Force );
  else
    throw std::logic_error( "ERROR in
DucFelyxObject::ApplyBrakingLoadcase : BoundConPtr not set!" );

  if ( noise > 0 )
    OutStream << "--< " << my_timer.elapsed()
    << " >-- " << " and node : " << distance( Nodes.begin(), nit ) << endl;
}


double DucFelyxObject::EvalMass( unsigned EleRefNrMin, unsigned
EleRefNrMax ) const {

  double mass = 0.0;
  PtrVector<element_type*>::const_iterator eleit;
  for ( eleit = Elements.begin(); eleit != Elements.end(); ++eleit ) {
    if ( ( *eleit ) ->GetRefNumber() >= EleRefNrMin && ( *eleit ) -
>GetRefNumber() <= EleRefNrMax )
      mass += ( *eleit ) ->EvalMass();
  }

  if ( noise > 0 )
    OutStream << "--< " << my_timer.elapsed()
    << " >-- " << "Mass of frame [kg] : " << mass * 1000 << endl;

  return mass * 1000;
}


// see next page
```

Delete all forces and moments and apply braking loadcase

Delete all consisting forces and moments

Apply Forces that introduce the braking moment

Find Node (785, 0.0, 44.0)to introduce the force

Define Boundary Condition Pointer

Gives back an error, if necessary

Output in the message window

Same for the second Node

End of ApplyBrakingLoadcase()

Eval mass of frame in kg

Only take elements in the specified range

Return mass in kg

End of EvalMass

```
double DucFelyxObject::EvalMaxStress( unsigned EleRefNrMin, unsigned
EleRefNrMax ) {

  EvalStresses();

  unsigned n = 0;
  float_type sb_max = 0.0, sax_max = 0.0, sn_max = 0.0, s_mises = 0.0,
s_mises_max = 0.0;

  PtrVector<StructElement*>::const_iterator eleit_begin, eleit;
  eleit_begin = Elements.begin();
  for ( eleit = eleit_begin; eleit != Elements.end(); ++eleit ) {

    if ( ( *eleit ) ->GetRefNumber() >= EleRefNrMin && ( *eleit ) -
>GetRefNumber() <= EleRefNrMax ) {


    for ( n = 0; n < ( *eleit ) ->Stresses.nrows(); ++n ) {

      sax_max = abs ( ( *eleit ) ->Stresses( n, 0 ) );


      sb_max = sqrt( pow( ( *eleit ) ->Stresses( n, 1 ), 2 ) +
                pow( ( *eleit ) ->Stresses( n, 2 ), 2 ) );

      sn_max = sax_max + sb_max;



      s_mises = sqrt( pow( sn_max, 2 ) + 3 * pow( ( *eleit ) ->Stresses( n, 3 ) ,
2 ) );

      if ( s_mises > s_mises_max )
        s_mises_max = s_mises;

      if ( noise > 1 ) {
        OutStream << "Stresses at node " << setw( 3 ) << n + 1
        << " of elem " << setw( 5 ) << distance( eleit_begin, eleit ) + 1
        << " : sax_max=" << setw( 13 ) << sax_max
        << " sb_max=" << setw( 13 ) << sb_max
        << " sn_max=" << setw( 13 ) << sn_max
        << " s_mises=" << setw( 13 ) << s_mises << endl;
      }
    }
   }
  }
  if ( noise > 0 )
    OutStream << "--< " << my_timer.elapsed()
    << " >-- " << "Maximum von Mises stress in frame [ N/mm2 ] : " <<
s_mises_max << endl;

  return s_mises_max;
}
```

Eval max stress of frame, in a range of element reference numbers

Eval stress vectors of elements

Loop through all elements

Only look at elements in certain range of element reference numbers

Loop through stress vectors of element

Eval axial stress -> using abs(s,ax) to always get a max stress

Eval max bending stress in model

Equals vector addition of s, b, v mas and s, b, z, max

Gives a total normal stress

Eval equivalent maximal stress after von Mises by including torsional shear stress

End of max. Stress

Section of DucFrameFEA_solution.cpp

This file is the virtual executable. The header file DucFelyx.h is included and as a result all member functions and attributes of StructObject are also included. Also all new functions like EvalMass are also imported.

```
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

// FELyX Main header
#include "DucFelyx_solution.h"

#include <boost/program_options.hpp>
namespace po = boost::program_options;

#include "boost/filesystem/operations.hpp"
#include "boost/filesystem/path.hpp"
namespace fs = boost::filesystem;

int main( int ac, char* av[] ) {

   DucFelyxObject FEM("filename.ansys", "path",0 );

   FEM.NodesReordering( bwalgo );

   FEM.updateTubeProperties(ParamPath,ParamFormat);

   // FEM.SaveAnsysModel();

   FEM.ApplyTorsionLoadcase();

   if ( solver == "skyline" )
     FEM.DirectSolver();
   else if ( solver == "pardiso" )
     FEM.SparseSolver();
   else
     std::cerr << "WARNING: No valid solver type specified, evaluating
nothing!\n ";

   double Stiffness= FEM.EvalTorsionStiffness();

   FEM.ApplyBrakingLoadcase();

   if ( solver == "skyline" )
     FEM.DirectSolver();
   else if ( solver == "pardiso" )
     FEM.SparseSolver();
   else
     std::cerr << "WARNING: No valid solver type specified, evaluating
nothing!\n ";

   double MaxStress= FEM.EvalMaxStress( 1, 2 ) ;

   double Mass= FEM.EvalMass(1,2);

   exportObjectives( ResPath, Mass, Stiffness, MaxStress);

 }
return 0;
}
```

Include program options lib from boost

Include filesystem lib from boost

Import FE Model

//StructObject FEM(ModelPath.leaf(), fs::complete(ModelPath).branch_path().native_file_string(),noise);

Save new model if needed

Torsional Loadcase

Get torsional stiffness

Braking Loadcase

Get max. equivalent stress (van Mises)

Get Mass

Save results

Output

- The complete source files are on the homepage
- Depending on the noise (definition of output: 0, 1 or 2 where 2 plots the complete FE simulation information) more or less information are printed
  - For noise = 1

```
FE analysis of Ducati steel trellis frame using FELyX
File Constructor of FelyxObject got called; start time logging
--<0>-- Loading Ansys Model from: /path/tutorial
--<0.02>-- Mass of frame [kg] 7.40774
--<0.02>-- Applied braking loadcase to node: 13
--<0.02>-- and node: 15
--<0.02>-- Link node DOF's to GSM index
--<0.02>-- Initialize GSM of size: 191
--<0.02>-- Evaluate ESM's and assemble them to GSM
--<0.03>-- Resize DofSolution vector and set its vals to zero
--<0.03>-- Eval load vector and apply inhomogeneous BC's
--<0.03>-- Start solving using direct sparse solver ...
--<0.03>-- Recordering completed ... nonzeros in factors = 7484
--<0.03>-- Factorization completed ... --> numnber of factorization MFLOPS = 0 --> to peak memory consumption = 7554
--<0.03>-- Solution done - status: 0
--<0.03>-- Postprocessing: Storing nodal deformations to nodes
--<0.03>-- Postprocess: Evaluate stresses
--<0.03>-- Maximum von Mises stress in frame [N/mm2]: 451.156
--<0.03>-- Applied torsion loadcase to node: 14
--<0.03>-- Link node DOF's to GSM index
--<0.03>-- Initialize GSM of size: 191
--<0.03>-- Evaluate ESM's and assemble them to GSM
--<0.04>-- Resize DofSolution vector and set its vals to zero
--<0.04>-- Eval load vector and apply inhomogeneous BC's
--<0.04>-- Start solving using direct sparse solver ...
--<0.04>-- Recordering completed ... nonzeros in factors = 7484
--<0.04>-- Factorization completed ... --> numnber of factorization MFLOPS = 0 --> to peak memory consumption = 7554
--<0.04>-- Solution done - status: 0
--<0.04>-- Postprocessing: Storing nodal deformations to nodes
--<0.04>-- Torsion stiffness of frame [Nm/degree]: 1313.86
--<0.04>-- Save FE-Model in ANSYS format to: /pathl/DucatiFrameSolution.ansys
FELyX analysis done
```

- This results can be used for further calculations or for an optimization
- The number in the <>-brackets is the CPU time