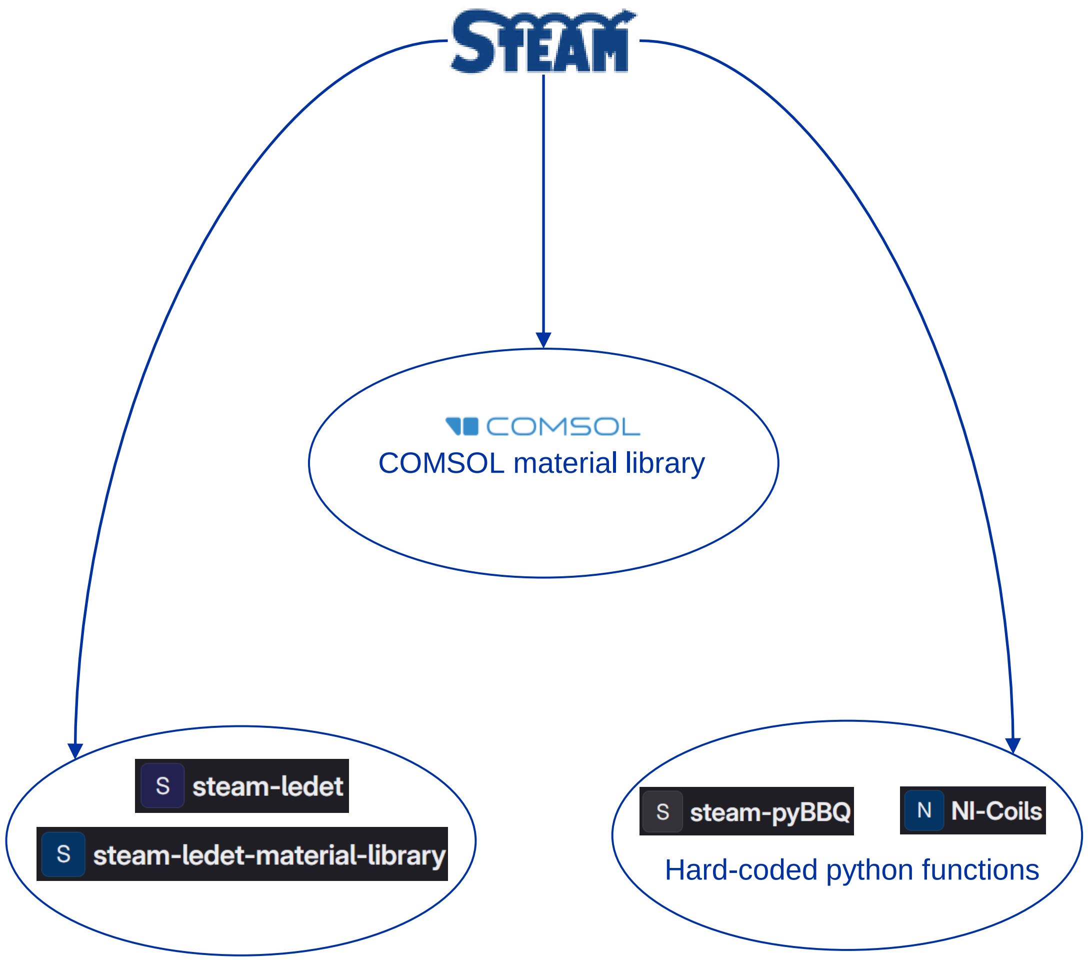
A Unified Common Source for Material Properties across Simulation modelling tools

Georgia Zachou

07/03/2024

Challenge

Modelling tools are written or even use **various programming language**. This means:



Independent individual sources and functions exist **for each simulation tool**.

Material functions may **slightly vary or even become invalid** in a specific ranges.

Potentially **different simulation results** for the **same phenomena**.

Reasons:
Lack of well-documented sources, different sources, mistakes while writing functions.

A few words for STEAM modelling tools…

CERNGetDP[1]/FiQuS[2]:

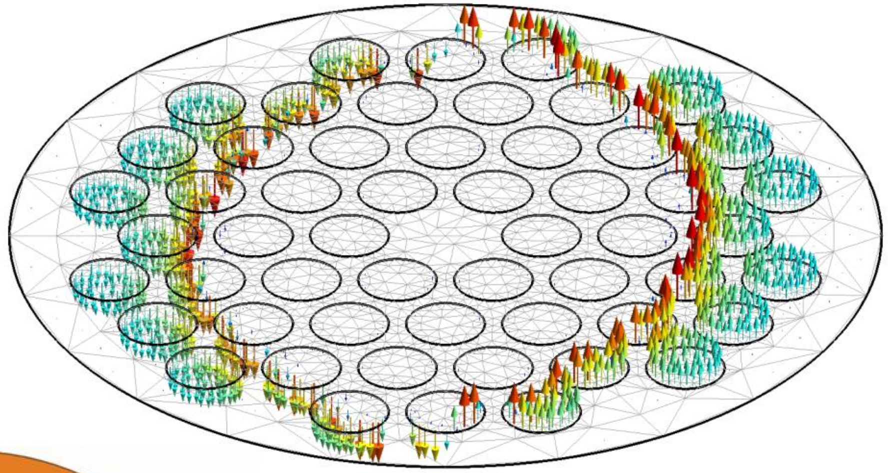
[1]*CERNGetDP GitLab repository*

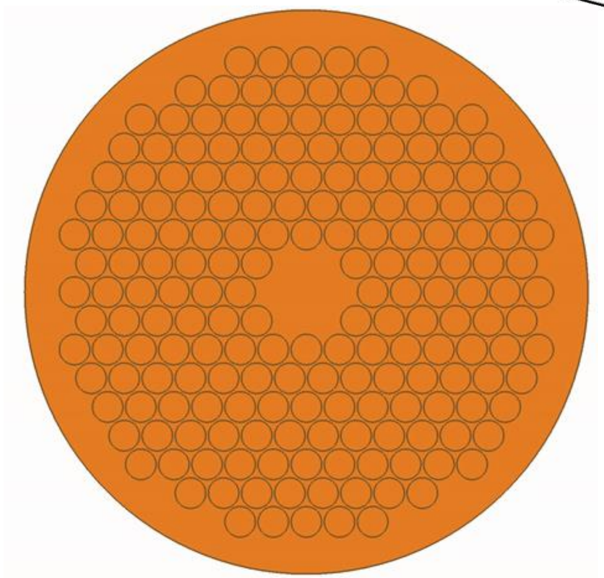
[1]*FiQuS GitLab repository*

Focus: Modeling 2D/3D thermo-electromagnetic transients in superconducting magnets and cables.

Language/Interface: FiQuS relies on Gmsh for geometry and meshing and on GetDP for solving and postprocessing.

Importance: Provides stable and accurate solutions for discrete problem solving. Used previously **hardcoded GetDP functions.**





A few words for STEAM modelling tools…

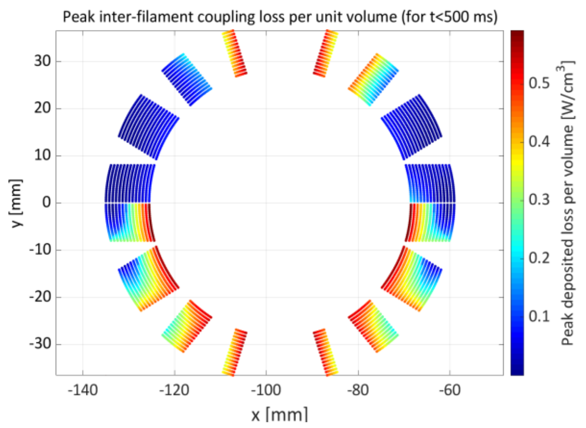
STEAM-LEDET[1]:

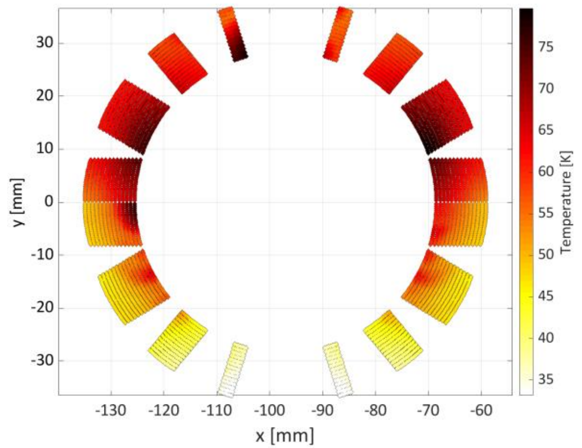
[1]LEDET GitLab repository

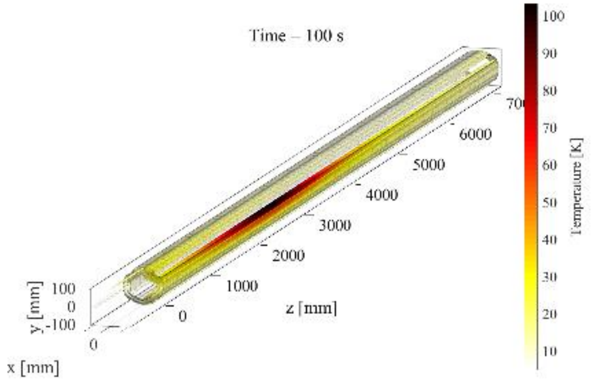
Focus: Modeling electromagnetic and thermal transients in magnets (**mainly LTS**, Niobium-Titanium, Niobium-Tin) in 2D & 3D geometries.

Language: Matlab.

Importance: Used for protection studies to assess magnet survival in failure scenarios. Used **steam-ledet-material-library**.







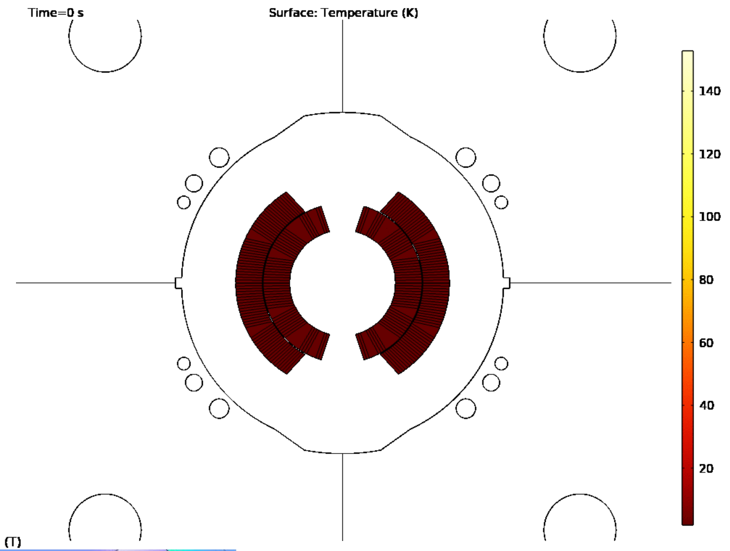
A few words for STEAM modelling tools…

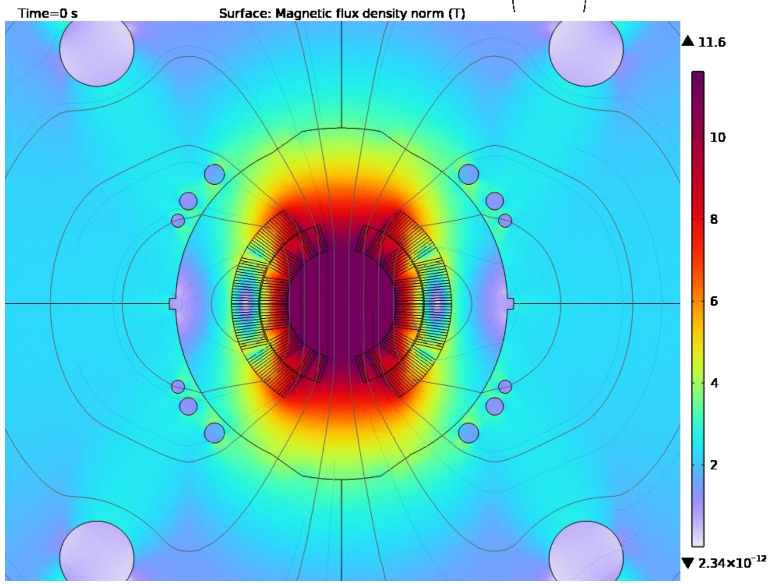
SIGMA:
[1]*Steam-SIGMA [GitLab repository]*

Focus: simulates electro-magnetic and thermal transients in superconducting magnets in a 2D geometry using COMSOL (FE model).

Language/Interface: Java, with a Python wrapper called pySIGMA.

Importance: Integrates steam-material-library into COMSOL simulations for accurate results. Used integrated COMSOL material functions.





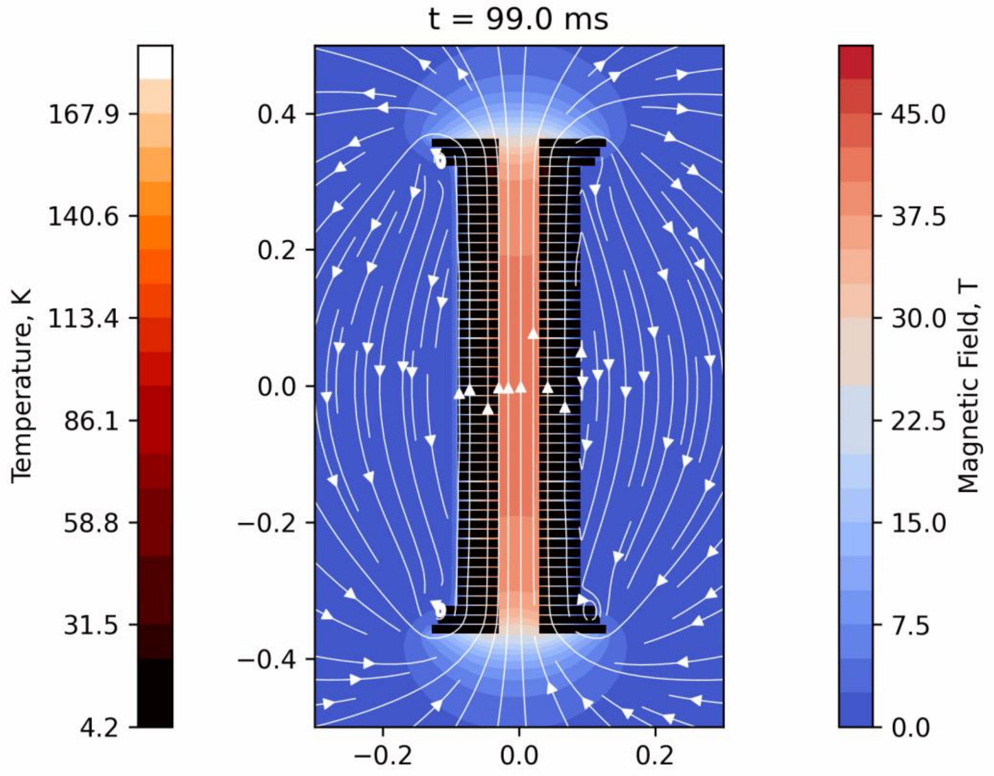
A few words for STEAM modelling tools…

NICQS[1]:
[1]*Ni-Coils [GitLab repository]*

Focus: Modeling thermo-electromagnetic behavior in non-insulated coils during operation and quench (mainly HTS).

Language/Interface: Python

Importance: Simulates thermal transients for High Temperature Superconductors (HTS), improving efficiency and accuracy. Used **hardcoded python functions**.



Steam-material-library as a solution…

Steam-material-library introduction:
Unified source containing all material properties used across modelling tools.

The functions are based on **extrapolated experimental data** and **literature**.

There is **Consistent naming** convention for functions to include property type and version.

The functions are written in **C programming language** adding extra advantages.

Why C?

- **Efficiency: potential speed improvement.**
- **Low-level language that can be compiled into others**
- **Easily integrates with other languages for broader use**

Expectations:

- **Compilation of the functions into other languages like Matlab and Python, ensuring compatibility across all modelling tools.**
- **Reduction in discrepancies and errors during rewriting or implementation.**
- **Faster results across simulations.**

Steam-material-library Contents

Steam-material-library includes material properties and calculations for HTS and LTS, serving the purpose of simulating transients.

Supported Materials	Properties	Available derivatives for materials:	Others…
• Ag • AgMg • Aluminium Alloys (7075/1350/6061/5085/2024/2014) • BeCu • Brass • BSCCO2212 • Cu • G10 • Hastelloy • He • In • Iron (BH) • Kapton • Nb_3Sn • NbTi • Steel(Stainless Steel) • Stycast • Titanium	• Volumetric heat capacity • Specific heat • thermal conductivity • Resistivity • Jc/Ic: Critical Current density (LTS & HTS)	• Ag • Aluminium Alloys (7075/1350/6061/5085/2024/2014) • BeCu • Brass • BSCCO2212 • Cu • G10 • Hastelloy • In • Kapton • Nb_3Sn • NbTi • Steel(Stainless Steel) • Titanium	• Current Sharing for HTS

Compilation Process

One single C source

to .dll → COMSOL

BBQ
SIGMA

wrapper → python

STEAM-materials python package[1] used in:
PyBBQ
NICQS

to .mex → MATLAB

STEAM_MatPro MatLab-mex class[2] used in:
LEDET

CERNGetDP → FiQuS

Material functions used in FiQuS through CERNGetDP[3]

[1]*Pypi steam-material-library*
[3]*STEAM_MatPro MatLab class*
[3]*CERNGetDP interface*

Mex compilation and use

1. All C functions are written based on a **specific template** so that:
   - mex are compiled properly
   - but also, COMSOL **requires** this structure to work.

2. **generic_wrapper.cpp** ensures 1 output argument C-functions are compiled into mex.
   - generic_wrapper_v2.cpp has been created to compile functions with more than 1 outputs, but not used as it requires different structure on the C template-which has an impact on COMSOL implementation.

3. The MatLab class **STEAM_MatPro.m** ensures that the input and output arguments of the mex files are **aligned precisely** with those of the MATLAB functions in STEAM-LEDET.
   Mex and STEAM_MatPro compilation is **automated** through a pipeline using a **virtual machine**, and they are also stored on GitLab as **artifacts zip** files.

4. Run through the MatLab **class** or **independently**.

```
#include <math.h>
#include <float.h>
#include <stdlib.h>
#include <string.h>
#ifdef _MSC_VER
#define EXPORT __declspec(dllexport)
#else
#define EXPORT
#endif

static const char *error = NULL;

EXPORT int init(const char *str) {
  return 1;
}

EXPORT const char * getLastError() {
  return error;
}

EXPORT int eval(const char *func,
                int nArgs,
                const double **inReal,
                const double **inImag,
                int blockSize,
                double *outReal,
                double *outImag) {

    int i;

    if (strcmp("C_function_name", func) == 0) {
```

```
STEAM_ML_mex = STEAM_MatPro('mex');
STEAM_ML_mat = STEAM_MatPro('m');

CvCu_mex = STEAM_ML_mex.handle_CvCu_nist(T);
CvCu_mat = STEAM_ML_mat.handle_CvCu_nist(T);
```

```
[CvCu, ~] = CFUN_CvCu_v1('CFUN_CvCu_v1', T);
```

GetDP implementation and use

1. **csv** with **function characteristics:**
   - **c_function_name**
   - **GetDP_function_name**
   - **Input_const_param**
   - **Input_var_param**
   - **mapping**
   - **in_helper**

```
c_function_name,GetDP_function_name,input_const_params,input_var_params, mapping,in_helper
CFUN_CvNbTi_v1,CFUN_CvNbTi_T_B,3,2,0-1-2-3-4,0
CFUN_CvNbTi_v1,CFUN_CvNbTi_T,4,1,0-1-2-3-4,0
CFUN_CvNbTi_v1,CFUN_CvNbTi_B,4,1,1-0-2-3-4,0
```

2. **steam-material-adder.py automatically** adds material functions to CERNGetDP **source code**, specifically to files:
   - STEAM_Mat_Lib_ProDefines.h
   - STEAM_Mat_Lib.h
   - F_STEAM_Mat_Lib.cpp

```
void F_CvNbTi_legacy(F_ARG) {
  if(A->Type != SCALAR || (A+1)->Type != SCALAR || (A+2)->Type != SCALAR)
    Message::Error("Wrong inputs for CvNbTi_legacy!");

  int nArgs = 5;

  // initialization (here instead of in initInput to omit a compiler warning)
  double **inReal = new double*[nArgs];
  double **inImag = new double*[nArgs];
  double* outReal = new double[blockSize];
  double* outImag = new double[blockSize];
  initInput(nArgs, blockSize, inReal, inImag);

  inReal[0][0] = (double)(A)->Val[0]; // T
  inReal[1][0] = (double)(A+1)->Val[0]; // B
  inReal[2][0] = (double)(A+2)->Val[0]; // I, calculated as I = J * Ic/Jc?
  inReal[3][0] = Fct->Para[0]; // C1
  inReal[4][0] = Fct->Para[1]; // C2


  int code = CvNbTi_legacy::eval("CFUN_CvNbTi", nArgs,
```

```
void F_CvNbTi_legacy(F_ARG) ;
```

```
{"CvNbTi_GetDP"     ,  (CAST)F_CvNbTi_legacy      ,  2,   3},
```

To add a C-function on GetDP source code one must **add its name and characteristics at the csv**, and the pipeline will update **automatically** the CERNGetDP source code.

DLL compilation and COMSOL implementation of steam-material-library

1. Update the .list files in the Compiler repository with the names of the functions to compile

Two methods:

1. Manually update the .list files.
2. Automatically update using the provided scripts:
   - make_list.py for Source_c repository.
   - make_list.py for Source_c_derivatives repository.
   - make_list.py for Source_c_old repository.

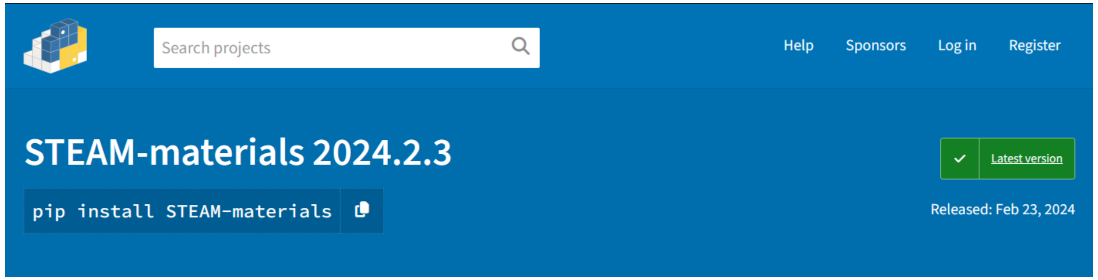
2. Run automaticLibraryCompiler.bat to compile functions listed in the .list files into DLLs:

A Batch script calls Microsoft Visual Studio for compilation. All function names are written into .list files. The script automates compilation through Microsoft Visual Studio:

- Automatically updates the list files.
- Compiles each source C file into a .obj file using the Microsoft C compiler (cl).
- Links the .obj file into a .dll using the Microsoft linker (link), creating the DLL.

Dual purpose:

1. **Integration with COMSOL multi-physics platform.**
2. **Serving as a PyPI library for Python-based tools.**

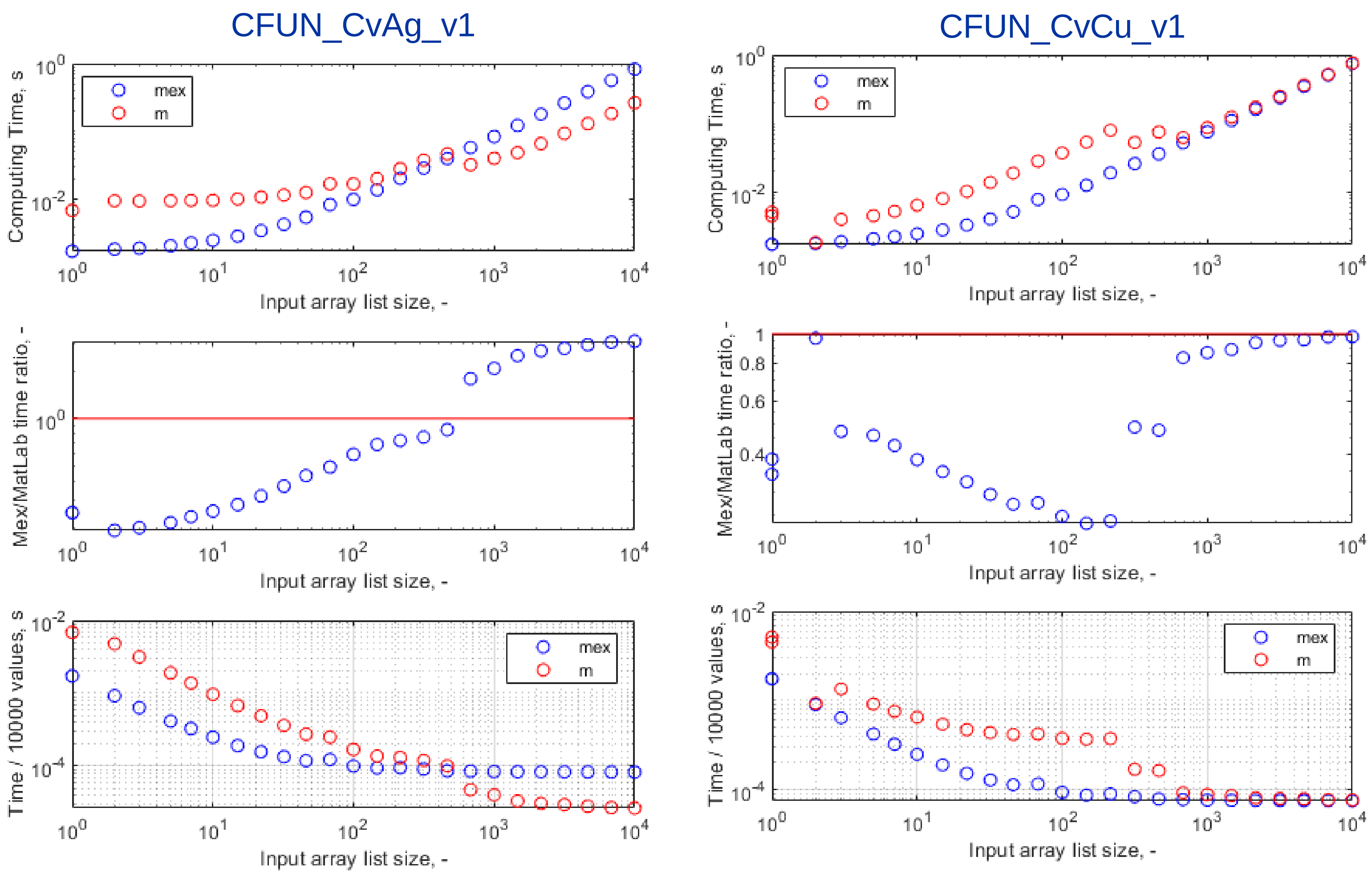


Tests

Two types of tests conducted on various versions of functions, **runtime** tests and **unit** tests.

- **Runtime Tests include comparison between:**
  - ➢ MatLab functions and newly generated Mex files.
  - ➢ STEAM-material functions from Pypi library and previously used (hard-coded) Python-based material functions.
- **Unit tests include comparison between:**
  - ➢ GetDP implemented functions and Python files from PyPI library tested and compared.
  - ➢ Previously used MatLab functions and newly generated Mex files are compared.

MatLab – Mex runtime tests



Subfigure (a): **computing time** vs**. input array size** for a generated Mex file and its corresponding MatLab function.

Subfigure (b): computing time is **lower** using the Mex file for input array sizes below **512 elements**.
Significant increase observed above 512 elements of input array.

Subfigure (c): **normalized** computing time (time per 10,000 values) for both Mex and MatLab implementations

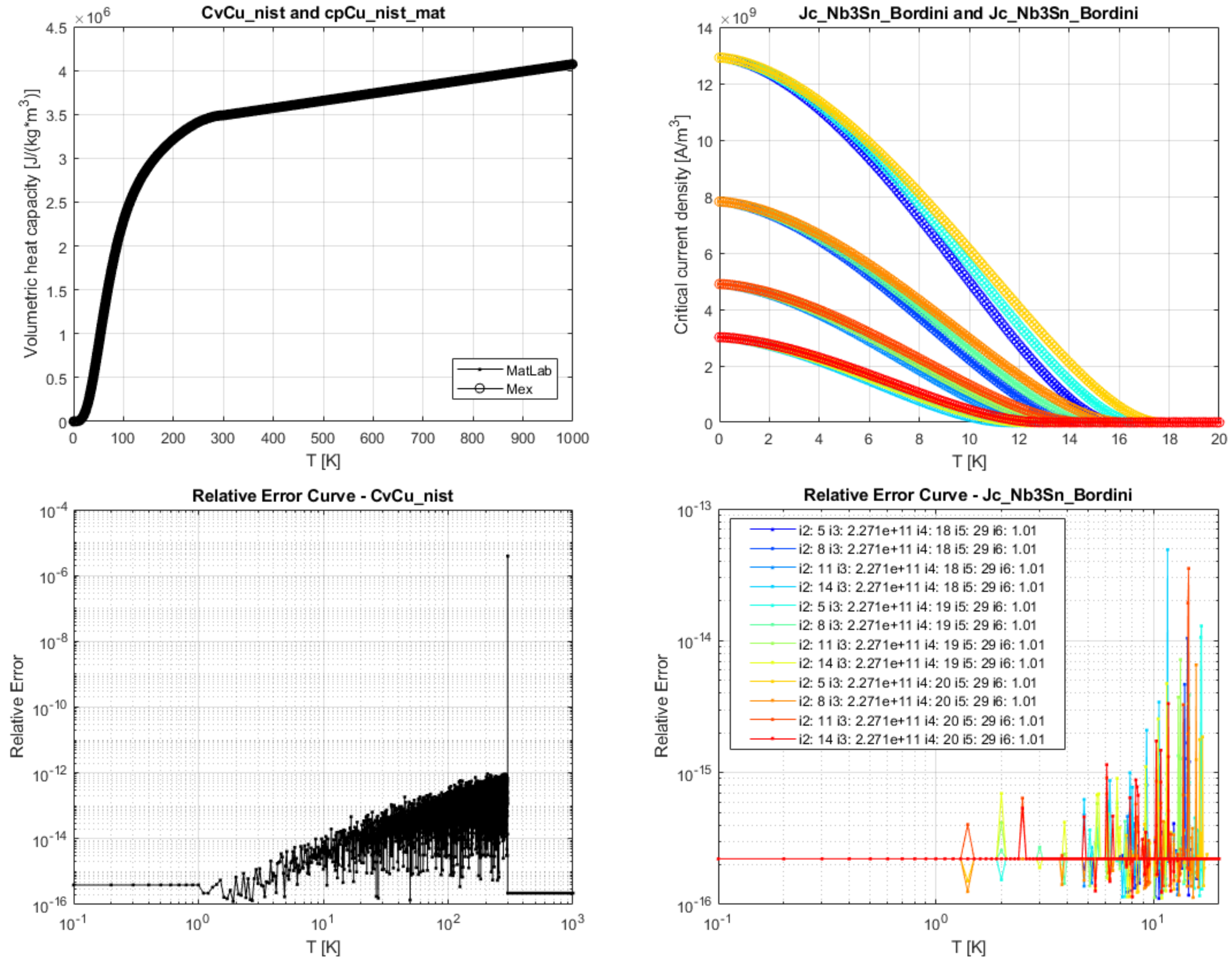
In every testing array size, the function is called 10,000 times.

Above 512: MatLab is doing some multithread magic…

MatLab – Mex unit tests

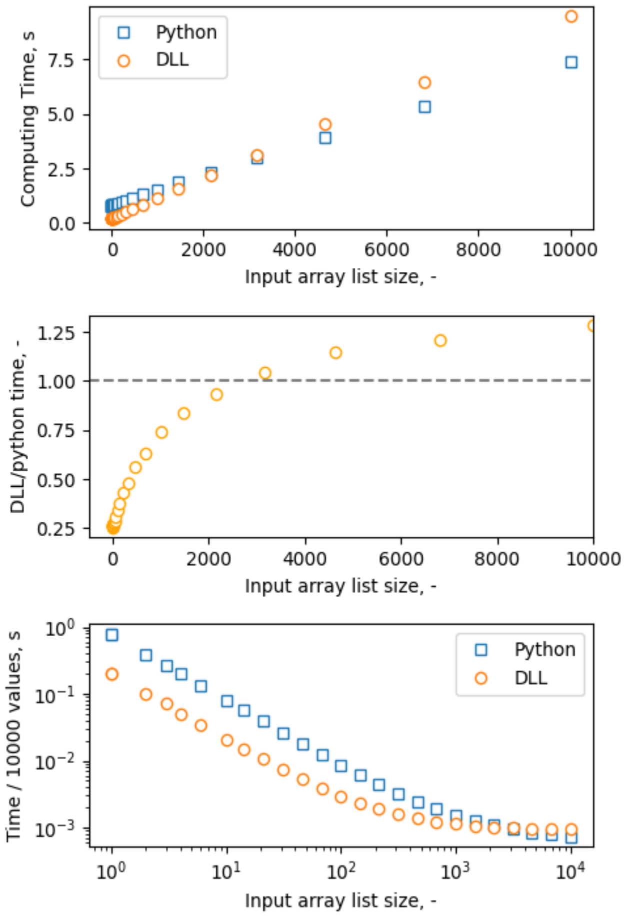
Subfigures above: MatLab and Mex functions are plotted against Temperature in Kelvin [K]. The two lines overlap.

Subfigures below:
Relative error of MatLab and Mex functions is plotted against Temperature in Kelvin [K]. Typically, small errors are observed around the scale of $\sim 10^{-16}$.

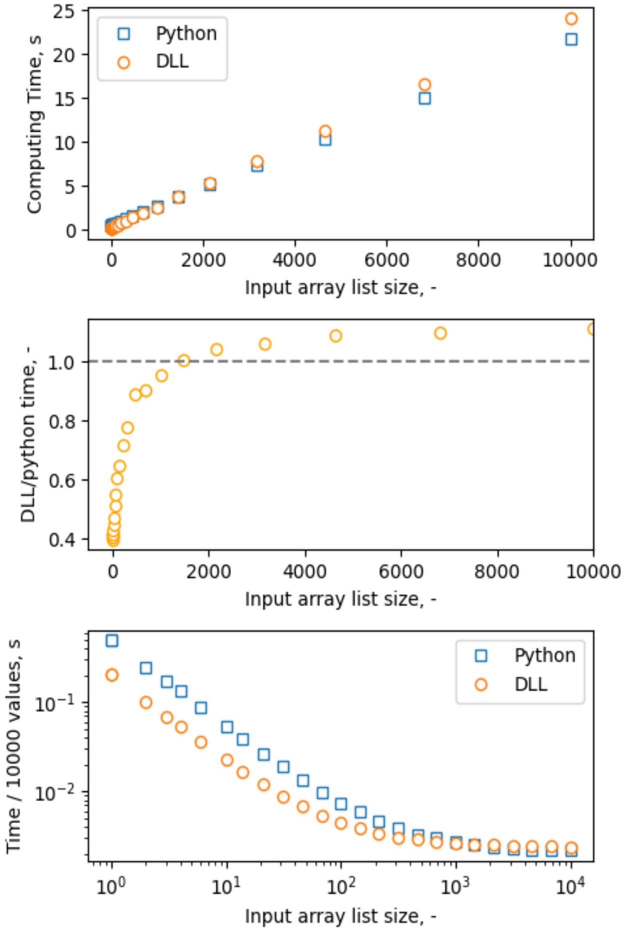


Runtime tests python-DLL

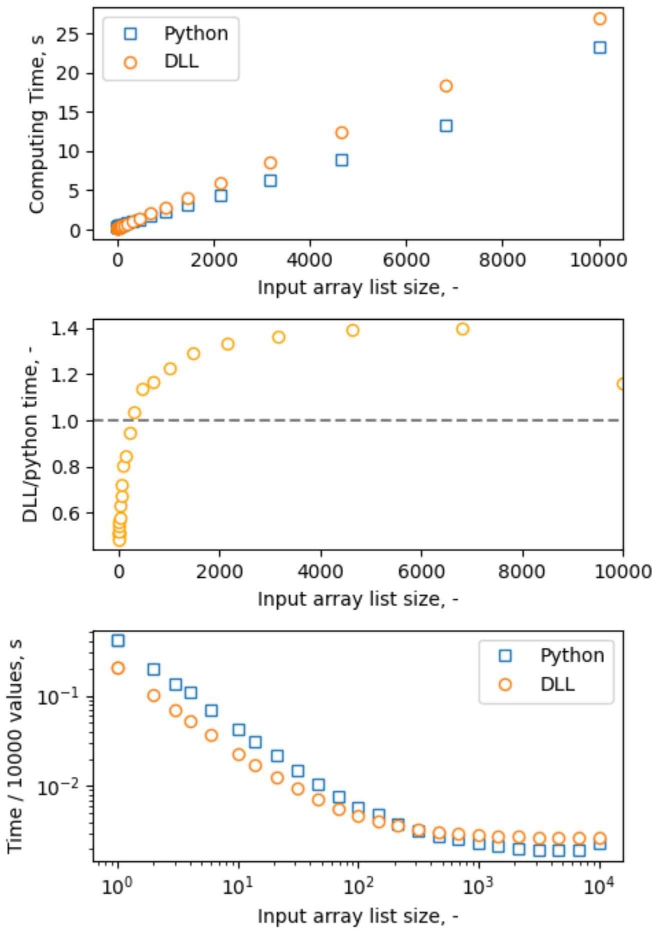
CFUN_CvCu_v1



CFUN_kStycast_v1



CFUN_rhoCu_v1



Same exact plots as the previous MatLab-Mex runtime comparison.

It seems that for small input array DLLs are faster than python hard-coded functions.

Runtime results

Function name	MatLab	Mex	STEAM-Materials	Python
CFUN_CvAg	0.3729	0.2721	0.9200	0.8399
CFUN_CvCu	1.0957	0.2451	0.9270	0.4815
CFUN_kG10	1.7469	0.2432	0.7659	0.8051
CFUN_rhoSS	0.1609	0.0771	0.2531	0.2810
CFUN_rhoCu	2.7372	1.7013	0.9762	1.2088

N = 500 (input array size) – 10,000 function calls

Potential speed improvement for most of the functions especially in MatLab-Mex comparison.

Function name	MatLab	Mex	STEAM-Materials	Python
CFUN_CvAg	1.4728	2.3916	6.1309	6.7191
CFUN_CvCu	2.7532	2.0656	3.0396	3.4927
CFUN_kG10	3.8893	2.1265	6.7812	5.2503
CFUN_rhoSS	0.7526	0.3201	1.4031	1.2822
CFUN_rhoCu	6.0165	17.2304	10.5938	6.6768

N = 5000 (input array size) – 10,000 function calls

Potential speed improvement for some functions.

CERNGetDP automated tests

make_tests.py is the main script for the GetDP tests.

- **It uses unittest PyPI library to make comparison on the DLL-GetDP functions results, considering a relative and absolute tolerance $\sim 10^{-4}$.**
- **The informations for the functions are taken from input_test.csv.**
- **It creates locally 3 folders:**
  - Outputs_msh: Contains GetDP msh files.
  - Outputs_pro: Contains GetDP pro files.
  - Outputs_txt: Where python results and GetDP results are stored in each column correspondingly.

The testing procedure is automated on a GitLab pipeline and the Outputs_msh, Outputs_pro and Outputs_txt are stored on GitLab Artifacts in a .zip file in case we need to check the output files of each function in detail.

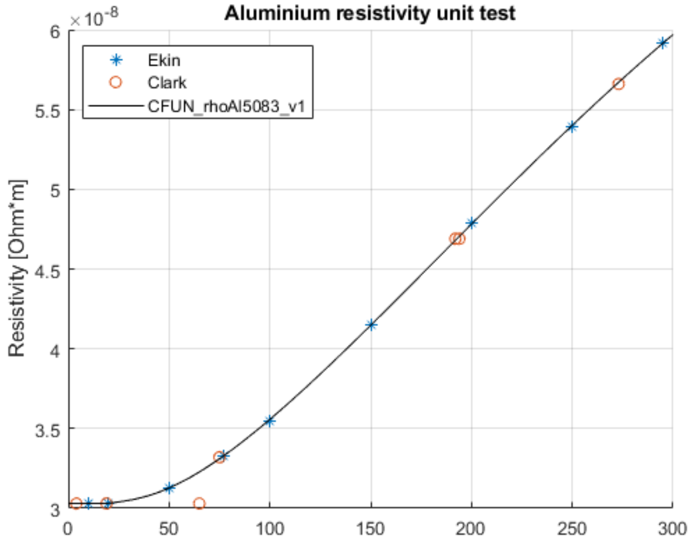
Aluminium functions & unit tests from literature

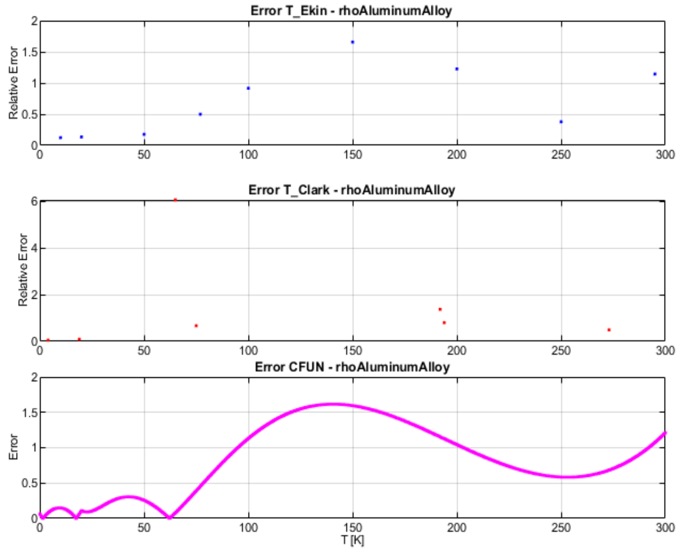
Based on existing literature, a comparison was conducted on aluminium functions. Following this research, a single function was developed to describe **each property** (e.g. Volumetric heat capacity, thermal conductivity, resistivity) of **all aluminium alloys based on their RRR** values.
For additional details, please refer to the GitLab repository.

Experimental data used in the study were obtained from the following sources:

- Ekin
- Clark
- NIST

This is an example testing procedure for future unit tests on the rest of the material functions…





STEAM Material Library

steam-material-library ☆1 0

Search

Home Properties Naming Installation and Use Tools Compiling Contributing Licensing Contact

Automated documentation - Website

template_md_files_properties.py script using **jinja2** templates, generates **automatically** script for the update of the website. **This update is required whenever there is a change concerning material functions structure or naming**.

1. The info for the documentation of the functions are taken from their naming and the main **CSV** file of the repository functionsNames_website_csv.
2. Using this info, template_md_files_properties.py **classifies** the functions based on their **properties**. It uses functions_md_files_templates.md template for writing the info in a form of documentation for each function straight to the website's source code.
3. The template also uses **figures** generated from MatLab-Mex unit tests to provide an illustration of the properties of the functions.
4. For the derivatives we have the same procedure.

STEAM Materials Library

Properties | Naming

Installation and Use | Tools | Compiling

Licensing | Contact

Thank you!

Mex compilation and use

generic_wrapper_v2

1. C functions with a specific template so that mex are compiled properly .. But also COMSOL.

2. Compiled into mex with generic_wrapper.cpp

3. The MatLab class STEAM_MatPro.m ensures that the input and output arguments of the mex files are aligned precisely with those of the MATLAB functions in STEAM-LEDET

```c
#include <math.h>
#include <float.h>
#include <stdlib.h>
#include <string.h>
#ifdef _MSC_VER
#define EXPORT __declspec(dllexport)
#else
#define EXPORT
#endif


static const char *error = NULL;

EXPORT int init(const char *str) {
  return 1;
}

EXPORT const char * getLastError() {
  return error;
}

EXPORT int eval(const char *func,
                int nArgs,
                const double **inReal,
                const double **inImag,
                int blockSize,
                double *outReal,
                double *outImag) {

    int i;

    if (strcmp("C_function_name", func) == 0) {
```

```matlab
STEAM_ML_mex = STEAM_MatPro('mex');
STEAM_ML_mat = STEAM_MatPro('m');

CvCu_mex = STEAM_ML_mex.handle_CvCu_nist(T);
CvCu_mat = STEAM_ML_mat.handle_CvCu_nist(T);
```

```matlab
[CvCu, ~] = CFUN_CvCu_v1('CFUN_CvCu_v1', T);
```

4. Run through the MatLab class or independently.

CERNGetDP material functions implementation

csv with function characteristics

↓

Steam-material-adder automatically adds material functions to CERNGetDP source code

```
c_function_name,GetDP_function_name,input_const_params,input_var_params, mapping,in_helper
CFUN_CvNbTi_v1,CFUN_CvNbTi_T_B,3,2,0-1-2-3-4,0
CFUN_CvNbTi_v1,CFUN_CvNbTi_T,4,1,0-1-2-3-4,0
CFUN_CvNbTi_v1,CFUN_CvNbTi_B,4,1,1-0-2-3-4,0
```

→ STEAM_Mat_Lib_ProDefines.h

```
void F_CvNbTi_legacy(F_ARG) {
  if(A->Type != SCALAR || (A+1)->Type != SCALAR || (A+2)->Type != SCALAR)
    Message::Error("Wrong inputs for CvNbTi_legacy!");

  int nArgs = 5;

  // initialization (here instead of in initInput to omit a compiler warning)
  double **inReal = new double*[nArgs];
  double **inImag = new double*[nArgs];
  double* outReal = new double[blockSize];
  double* outImag = new double[blockSize];
  initInput(nArgs, blockSize, inReal, inImag);

  inReal[0][0] = (double)(A)->Val[0]; // T
  inReal[1][0] = (double)(A+1)->Val[0]; // B
  inReal[2][0] = (double)(A+2)->Val[0]; // I, calculated as I = J * Ic/Jc?
  inReal[3][0] = Fct->Para[0]; // C1
  inReal[4][0] = Fct->Para[1]; // C2


  int code = CvNbTi_legacy::eval("CFUN_CvNbTi", nArgs,
```

→ STEAM_Mat_Lib.h

```
void F_CvNbTi_legacy(F_ARG) ;
```

→ F_STEAM_Mat_Lib.cpp

```
{"CvNbTi_GetDP"      ,  (CAST)F_CvNbTi_legacy       ,   2,    3},
```