Constrained Optimization

ConstrainedOptimization is a UG4 Plugin that provides various algorithms used in a constrained optimization setting.

This document should give a thorough overview on ConstrainedOptimization combined with some usage examples. For first time users, using the plugin is not necessarily straightforward, as many things have to be considered. Especially the required data format of the model output and the optimization solver setup itself may present issues. But usually all these issues can be resolved after the user has grasped the gist of how the plugin is supposed to work.
Therefore, this manual is focused more on the usage of ConstrainedOptimization from a user perspective. For an explanation of how the plugin works internally, you may refer to the doxygen file supplied with the plugin.

Table of Contents

Installation

The newest version of the plugin can be downloaded from https://github.com/CowFreedom/ConstrainedOptimization

While compilation of ConstrainedOptimization has no dependencies, it assumes that the terminal command

```
$ ugshell
```

exists and starts a Lua compiler. This is usually satisfied by installing UG4 along with the corresponding environment variables. Experienced users could theoretically not install UG4 and set the environment variable *ugshell* to refer to any Lua (or even e.g. Python compiler). The rest of this manual assumes, however, that UG4 has been properly installed.

Adding ConstrainedOptimization to UG4

The installation equals the standard process for adding UG4 plugins described on the ughub GitHub page. Due to the usage of OS specific fgunctions for process generation, the plugin is not necessarily multiplatform. In its current form, the package has been shown to work on Windows 10, Linux Ubuntu, Rasperry Pi OS, Arch Linux, and macOS Mojave 10.14.4. On Windows 10, two compilers were tested, GCC 9.3.0 and Visual Studio v16.3.10. It is assumed that the package also works for older compiler versions that support C++17. On Linux Ubuntu, only GCC was tested. On macOS Mojave the Clang compiler was used. Depending on your version of GCC and Clang, special flags might need to be set during the build process. This is explained below. The utilized Visual Studio compiler has not exhibited such necessities but older versions might.

GCC

Follow the steps on the ughub GitHub page. If errors occur, proceed with this text. The plugin makes use of C++ std::threads. This might necessitates activating the -pthread flag to the build process for ug4. Within your UG4 install library, go to

```
cd ug4/ugcore/cmake/
```

and open

```
ug_includes.cmake
```

Now search for

```
elseif("${CMAKE_CXX_COMPILER_ID}" STREQUAL "GNU")
  add_cxx_flag("-Wall")
  add_cxx_flag("-Wno-multichar")
  add_cxx_flag("-Wno-unused-local-typedefs")
  add_cxx_flag("-Wno-maybe-uninitialized")
```

and add

```
add_cxx_flag("-pthread")
```

to the GNU include statements. Now rebuild UG4 as described on the ughub GitHub page. The plugin should now be installed without any issues.

Clang

Follow the steps on the ughub GitHub page.. If errors occur, proceed with this text. The plugin makes use of C++11 features, like std::threads and constexpr. This might necessitates activating the -std=c++11 flag to the build process for ug4. Within your UG4 install library, go to

```
cd ug4/ugcore/cmake/
```

and open

```
ug_includes.cmake
```

Now search for

```
elseif("${CMAKE_CXX_COMPILER_ID}" STREQUAL "Clang")
    add_cxx_flag("-Wall")
    add_cxx_flag("-Wno-multichar")
```

and add

```
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")
```

to the Clang include statements. This ensures that the -std=c++11 flag is only added to the C++ compiler and not the C compiler in the build process. Now rebuild UG4 as described on the ughub GitHub page. The plugin should now be installed without any issues.

Descripton

The following algorithms are currently implemented:

Name	Description
Newton-Gauss	The Gauss Newton procedure minimizes functions that can be represented as a sum of squares.
Particle Swarm Optimization	The Particle Swarm Optimization (PSO) algorithm minimizes any continuous function on a bounded domain.
Geometry Sampler	Takes random samples from an arbitrary loss function.

The inner workings of each of these algorithms are either self explaining or can be found in publically available lecture. The main feature of **ConstrainedOptimization** is the compartmentalization of problem formulation, solution finding and the underlying computation mechanism. A problem is formulated via child instantiations of the Evaluation class type, which includes details such as the objective function to be optimized or problem specific parsing details. Solution finding algorithms are represented as classes such as NewtonOptimizer. Computational interaction with UG4 is abstracted away in instances of ComputationMode. Combined, this setup gives a flexible and modular platform that can be adapted for many problems at hand. For a descripton of the inner workings of this plugin, please refer to the doxygen file supplied with the plugin.

Working example: Fitting disease data to an epidemics model

The following example fits Covid-19 disease data to a odinary and partial differential equations model used for epidemiological purposes. First, the real world dataset is presented. Then the epidemiological model trying to explain the real life dataset is described. Then, the data output of the epidemiological model is rearragned into a format that ConstrainedOptimization can understand.

Afterwards, all the necessary files for an optimization of parameters through ConstrainedOptimization are created and the optimization algorithms are run. Lastly, results are discussed. For the entire paper where this exact scheme was used please refer to [Covid Paper].

Note: The model described in the [Covid Paper] requires UG4 software suite to be configured with ConvectionDiffusion and LIMEX plugins.

Dataset

The real world data set can be found in the CollectedData directory in the parameteroptimization folder.

For the PDE model the real world dataset comprises of one file which contains Covid-19 data for 7 cities spanning a total of 40 days.

The ODE models from the Epidemics plugin use a dataset spanning a time period of 30 days and are aggregated values for a larger region.

Source for both the datasets:

https://github.com/jgehrcke/covid-19-germany-gae

Epidemiological Model

Both the ODE and the PDE models are based on compartamental mathematical models. These models divide the population in, as the name suggests, compartments and describes the dynamics of the population flow from one compartment to another.

5 Compartments:

- Susceptibles
- Exposed
- Infected
- Recovered
- Deceased

The flow of the population between the compartments is governed by parameters.

Parameters:

- **alpha** = Rate of infection. This parameter is responsible for the flow from Suceptibles to Exposed.
- **Kappa** = Rate of transission between Exposed and Infected.
- **qq** = Incubation time. This determines how long a person would stay in the Exposed group.
- **Pp** = Duration of sickness.
- **Theta=** Mortalitiy rate.

The Models which can be found in the Epidemics plugin are sovled using Runga-Kutta-4 or a linear implicit solver. The PDE formulation is discretized using Finite Difference.

The model from the [Covid Paper] was spatially discretized using Finite Volumes and temporally discretized with LIMEX.

These parameters are essential for the understanding and investigation of the dynamics of the spread of the disease.

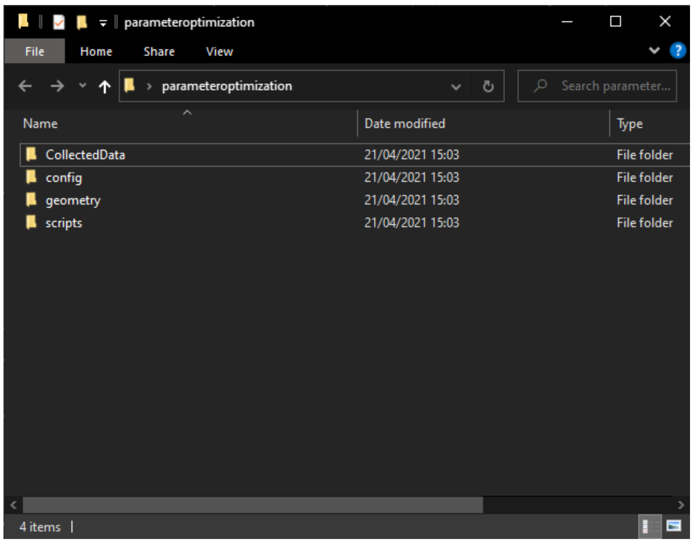
Optimization with Constrained Optimization

The optimization is a integral part of working with models. The parameters of the epidemiological models need to be fitted to the real world data. This allows researchers to test their models' correctness. Once decent values have been found, the epidemiological models can be used to forcast the spread of the disease, which is of great interest to policy makers.

Constrained Optimization reads the real data set and using different solvers tries to minimize the objection function (Sum of squared errors, in this case). Hence the output of the model needs to be in the same format as the real world data.

The optimization process of using CO is as follows:

1. Add the real dataset to the model folder which is going to be optimized. The first column of the file must be time and the first row of the file must start with ‚#' (column headers)-



Name	Date modified	Type
CollectedData	21/04/2021 15:03	File folder
config	21/04/2021 15:03	File folder
geometry	21/04/2021 15:03	File folder
scripts	21/04/2021 15:03	File folder

Figure 1: CollectedData contain the real world dataset. The other folders contain the required files for the epidemics model

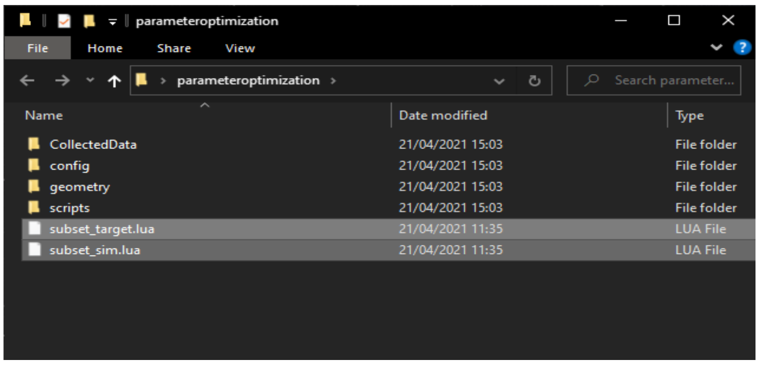
2. Depending on the model at hand, post processing of the model out put is required. The [Covid Paper] model required heavy post processing, where as the the models in the Epidemics plugin do not.

[TODO : insert image of unformated vs formated output]

3. Then subset_sim.lua and subset_targe.lua are added. It must be noted that the column 0 in both the files refers to the time column.

> <u>subset_sim.lua</u>
>
> This file describes the output files of the model and the columns of interest.
>
> <u>subset_target.lua</u>
>
> This file specifices the location and the columns of interest from the real world data.

The columns are read sequentially. So the order of the columns of the subset_target and subset_sim must correspond to each other. For dealing with multiple files for the real world data or output of the models, examples can be found in the Example folder. The idea remains the same, each colunm is read sequentally, one file after the next.



Name	Date modified	Type
CollectedData	21/04/2021 15:03	File folder
config	21/04/2021 15:03	File folder
geometry	21/04/2021 15:03	File folder
scripts	21/04/2021 15:03	File folder
subset_target.lua	21/04/2021 11:35	LUA File
subset_sim.lua	21/04/2021 11:35	LUA File

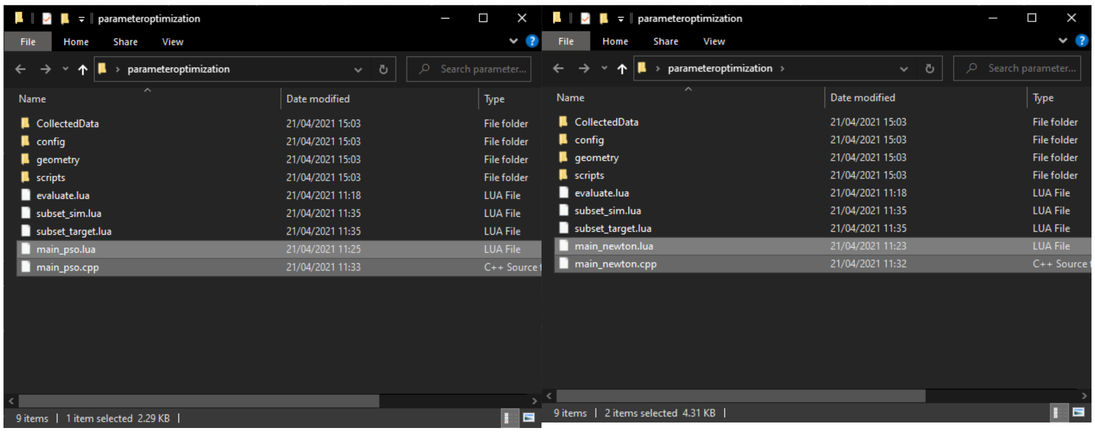
Figure 2: subset_sim and subset_target go in the working directory

4. Then the evaluate.lua is added. This file contians all the logic for the epidemics model with only a minor change. The parameters are loaded in via the output of Constrained Optimization iterations, and epidemics can be executed with new vlaues for the parameters.

Evaluate.lua can be created in two ways. Either the evaluate.lua contians all the logic of the model, or once the parameters have been loaded, calls the script which executes the model. Both examples can be found in the Examples folder.

5. The last step that remains is creating a main file that exectues and sets a few options for the solver chosen for the optimization procedure.

<u>Note:</u> main.lua OR main.cpp are required not both.



parameteroptimization

Name	Date modified	Type
CollectedData	21/04/2021 15:03	File folder
config	21/04/2021 15:03	File folder
geometry	21/04/2021 15:03	File folder
scripts	21/04/2021 15:03	File folder
evaluate.lua	21/04/2021 11:18	LUA File
subset_sim.lua	21/04/2021 11:35	LUA File
subset_target.lua	21/04/2021 11:35	LUA File
main_pso.lua	21/04/2021 11:25	LUA File
main_pso.cpp	21/04/2021 11:33	C++ Source

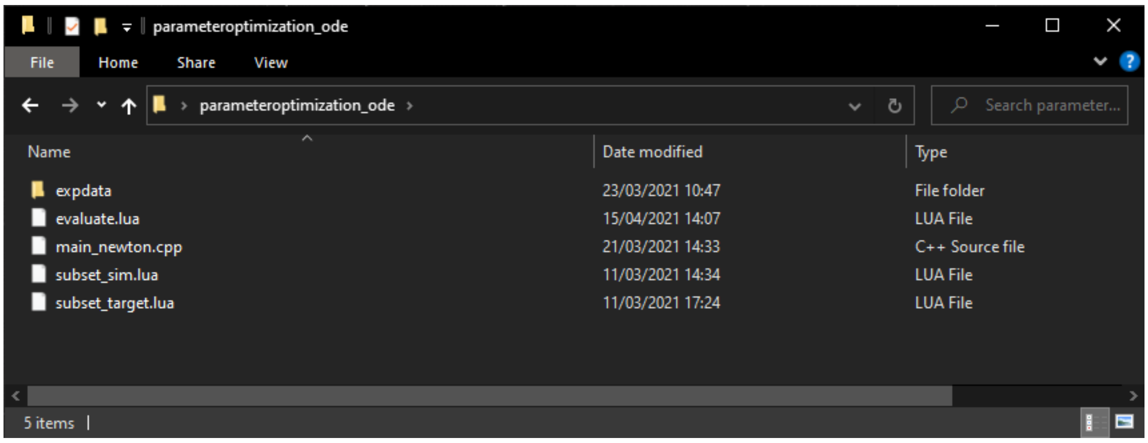
9 items | 1 item selected 2.29 KB |

parameteroptimization

Name	Date modified	Type
CollectedData	21/04/2021 15:03	File folder
config	21/04/2021 15:03	File folder
geometry	21/04/2021 15:03	File folder
scripts	21/04/2021 15:03	File folder
evaluate.lua	21/04/2021 11:18	LUA File
subset_sim.lua	21/04/2021 11:35	LUA File
subset_target.lua	21/04/2021 11:35	LUA File
main_newton.lua	21/04/2021 11:23	LUA File
main_newton.cpp	21/04/2021 11:32	C++ Source

9 items | 2 items selected 4.31 KB |

Figure 3: evaluate.lua and the main files also go in the working directory



parameteroptimization_ode

Name	Date modified	Type
expdata	23/03/2021 10:47	File folder
evaluate.lua	15/04/2021 14:07	LUA File
main_newton.cpp	21/03/2021 14:33	C++ Source file
subset_sim.lua	11/03/2021 14:34	LUA File
subset_target.lua	11/03/2021 17:24	LUA File

5 items |

Figure 4: The ODE model is defined in Epidemics plugin. In evalute.lua, by calling the right functions the model can be run. This figure, like Figure 3 shows a complete setup for a optimization problem.

FAQs‘