Multi-threaded Optimization Toolbox

Release 0.9.4

Robbert Harms

Dec 13, 2019

Contents

Chapter 1

Introduction

The Multi-threaded Optimization Toolbox (MOT) is a library for parallel optimization and sampling using the OpenCL compute platform. Using OpenCL allows parallel processing using all CPU cores or using the GPU (Graphics card). MOT implements OpenCL parallelized versions of the Powell, Nelder-Mead Simplex and Levenberg-Marquardt non-linear optimization algorithms alongside various flavors of Markov Chain Monte Carlo (MCMC) sampling.

For the full documentation see: https://mot.readthedocs.org

1.1 Can MOT help me?

MOT can help you if you have multiple small independent optimization problems. For example, if you have a lot of (>10.000) small optimization problems, with ~30 parameters or less each, MOT may be of help. If, on the other hand, you have one big optimization problem with 10.000 variables, MOT unfortunately can not help you.

1.2 Example use case

MOT was originally written as a computation package for the Microstructure Diffusion Toolbox, used in dMRI brain research. In diffusion Magnetic Resonance Imaging (dMRI) the brain is scanned in a 3D grid where each grid element, a *voxel*, represents its own optimization problem. The number of data points per voxel is generally small, ranging from 30 to 500 datapoints, and the models fitted to that data have generally somewhere between 6 and 20 parameters. Since each of these voxels can be analyzed independently of the others, the computations can be massively parallelized and hence programming in OpenCL potentially allows large speed gains. This software toolbox was originally built for exactly this use case, yet the algorithms and data structures are generalized such that any scientific field may take advantage of this toolbox.

For the diffusion MRI package *MDT* to which is referred in this example, please see https://github.com/cbclab/MDT.

1.3 Summary

- Free software: LGPL v3 license
- Interface in Python, computations in OpenCL
- Implements Powell, Nelder-Mead Simplex and Levenberg-Marquardt non-linear optimization algorithms
- Implements various Markov Chain Monte Carlo (MCMC) sampling routines
- Tags: optimization, sampling, parallel, opencl, python

1.4 Links

- Full documentation: https://mot.readthedocs.org

- Project home: https://github.com/cbclab/MOT
- PyPi package: PyPi

1.5 Quick installation guide

The basic requirements for MOT are:

- Python 3.x
- OpenCL 1.2 (or higher) support in GPU driver or CPU runtime

Linux

For Ubuntu >= 16 you can use:

- `sudo add-apt-repository ppa:robbert-harms/cbclab`
- `sudo apt update`
- `sudo apt install python3-pip python3-mot`
- `sudo pip3 install tatsu`

For Debian users and Ubuntu < 16 users, install MOT with:

- `sudo apt install python3 python3-pip python3-pyopencl python3-devel`
- `sudo pip3 install mot`

Mac

- Install Python Anaconda 3.* 64bit from https://www.continuum.io/downloads>
- Open a terminal and type `pip install mot`

Windows For Windows the short guide is:

- Install Python Anaconda 3.* 64bit from https://www.continuum.io/downloads
- Install or upgrade your GPU drivers
- **Install PyOpenCL using one of the following methods:**
    1. Use a binary, for example from http://www.lfd.uci.edu/~gohlke/pythonlibs/#pyopencl or;
    2. **Compile PyOpenCL with `pip install pyopencl`, this requires:**
        - Visual Studio 2015 (Community edition or higher) with the Python and Common Tools for Visual C++ options enabled
        - OpenCL development kit (NVidia CUDA or Intel OpenCL SDK or the AMD APP SDK)
- Open a Anaconda shell and type: `pip install mot`

For more information and for more elaborate installation instructions, please see: https://mot.readthedocs.org

1.6 Caveats

There are a few caveats and known issues, primarily related to OpenCL:

- Windows support is experimental due to the difficulty of installing PyOpenCL, hopefully installing PyOpenCL will get easier on Windows soon.
- GPU acceleration is not possible in most virtual machines due to lack of GPU or PCI-E pass-through, this will change whenever virtual machines vendors program this feature. Our recommendation is to install Linux on your machine directly.

Chapter 2

Installation

2.1 Ubuntu / Debian Linux

Using the package manager, installation in Ubuntu and Debian is relatively straightforward.

For **Ubuntu >= 16** the MOT package can be installed with a Personal Package Archive (PPA) using:

```
$ sudo add-apt-repository ppa:robbert-harms/cbclab
$ sudo apt update
$ sudo apt install python3-mot
```

Using such a PPA ensures that your Ubuntu system can update the MOT package automatically whenever a new version is out. For **Debian**, and **Ubuntu < 16**, using a PPA is not possible and we need a more manual installation. Please install the dependencies (*python3*, *pip3* and *pyopencl*) first:

```
$ sudo apt install python3 python3-pip python3-pyopencl python3-devel
```

and then install MOT with:

```
$ sudo pip3 install mot
```

After installation please continue with testing the installation below.

2.2 Mac

Installation on Mac is pretty easy using the Anaconda 4.2 or higher Python distribution. Please download and install the Python3.x 64 bit distribution, version 4.2 or higher which includes PyQt5, from Anaconda and install it with the default settings.

Afterwards, open a terminal and type:

```
$ pip install mot
```

To install MOT to your system.

2.3 Windows

The installation on Windows is a little bit more convoluted due to the lack of a package manager. The installation is a multi-step procedure:

1. Installing a *Python interpreter*
2. Installing the *OpenCL drivers*
3. Installing the *Python OpenCL bindings PyOpenCL*
4. *Installing MOT*

Installing Python

Since MOT is a Python package we need to install a Python interpreter. Considering that Python2 is soon end-of-life, this package only supports Python3.

The easiest way to install Python3 is with the Anaconda Python distribution. Please download and install the Python3.x 64 bit distribution, version 4.2 or higher which includes PyQt5, from Anaconda and install it with the default settings. If you are following this guide with the intention of installing MDT afterwards, please note that Anaconda versions prior to 4.2 have the (deprecated) PyQt4 as its Qt library. This is not a problem for MOT per se. However if you want to install MDT and use its Qt5 GUI, or more generally want to use Qt5 and packages that depend on Qt5, you will find benefit from installing Anaconda > 4.2 with PyQt5. If you insist on using an older Anaconda install or PyQt4 environment (also consider creating a new PyQt5 compatible env), note that this is possible, but you will have to install a PyQt5 package yourself, such as the m-labs PyQt5 Anaconda package and deal with its version conflicts, e.g. python version <= 3.4.

After installation type `Anaconda Prompt` in the Windows start bar and start the Anaconda Prompt command line interface.

Installing OpenCL drivers

To run OpenCL applications you need an OpenCL driver for your platform. Please download and install the correct device driver (Intel/AMD/NVidia) for your device with support for OpenCL 1.2 or higher. For graphics cards, make sure you are using the latest version of your graphics driver. For Intel processors download the OpenCL runtime from https://software.intel.com/en-us/articles/opencl-drivers (OpenCL Runtime for Intel Core and Intel Xeon Processors; towards the end). Note that installing the Intel driver is needed if you want to run OpenCL on your Intel CPUs. Is is not needed if you only want to run on your GPUs. As a rule, you need to have an OpenCL driver or runtime installed for every device you want to run computations on. Most often, having both CPU and GPU available is desirable.

Installing PyOpenCL

With the drivers installed and everything up to date, we can now proceed with installing the Python OpenCL bindings, `pyopencl`. This is often the most problematic step and errors later on (e.g. in testing MOT) often come down to an incomplete (failed) or incompatible (successful but not working) pyopencl package install. PyOpenCL can either be installed from a downloadable binary or be compiled from source. Using the binary is easiest since manual compilation is more difficult.

Alternative 1: Using a binary PyOpenCL package

Installing a precompiled binary wheel (.whl) is the easiest way to install PyOpenCL, but only works if the wheel is compiled for your specific Python implementation. At Christoph Gohlke website (http://www.lfd.uci.edu/~gohlke/pythonlibs/#pyopencl) you can find a range of PyOpenCL binary packages. If there is a compatible one for your system, download that version. You can see if it is compatible if the Python version and OpenCl version in the binary name matches that of your installed Python and driver supported versions. Note that many drivers, such as nVIDIA's only support OpenCL 1.2, so in that case take the wheel with "+cl12" in the name, and not e.g. "+cl21". For example if you have 64-bit Windows system with Python 3.5 and your GPU or CPU drivers support OpenCL 1.2 you need to download the wheel with "+cl12", `win-amd64` and cp35m in the name (note the format, cp<version>m, the m is important). (To check which Python version you have you can run `python --version` in the command line).

If there is no compatible version for your system to be found on Gohlke's website, here is a mirror of an older version by Gohlke that is compatible with most Python 3.5 systems: `pyopencl-2015.2.4-cp35-none-win_amd64`.

After the download, open an Anaconda Prompt (or a normal Windows cmd) as administrator (right-click the command and select "Run as administrator") and change directory to where you downloaded the `.whl` file. Then, install the binary using pip:

```
> cd %UserProfile%\Downloads
> pip install <filename>.whl
```

Please make sure you are in the right directory and please substitute `<filename>` for your downloadeded filename.

To test if this binary package works, open a Python shell (for instance by typing `python` in your open prompt) and type:

```
>>> import pyopencl
```

If that works (python >>> prompt reappears without messages about missing dll's and cffi problems), you are good to go. (exit the python prompt by typing `exit()` or Ctrl-Z then Enter) If you encounter an error that ends on something like:

```
> ImportError: DLL load failed: The specified procedure could not be found.
```

Then the binary package (.whl file) is not compatible with your OS version and/or Python installation. Either try a different wheel, or try the compilation procedure below.

Alternative 2: Compile PyOpenCL with Visual Studio 15

Installing `pyopencl` with pip from source code requires Visual Studio 2015 and an OpenCL SDK (this is different from a driver or runtime, the SDK includes compilation header files) to be present on your system. First, install Visual Studio 2015 with a few specific options enabled (under "Custom" during the installation):

- **[] Programming Languages**
  - **[] Visual C++**
    - [X] Common Tools for Visual C++ 2015
  - [X] Python Tools for Visual Studio

If you already have Visual Studio 2015 installed and are unsure if these options are enabled, you can rerun the installer to update your installation with additional options.

After this installation please download and install an OpenCL software development kit (SDK) matching the vendor of your graphics card or processor:

- For Intel, see https://software.intel.com/en-us/intel-opencl
- For AMD, see https://github.com/GPUOpen-LibrariesAndSDKs/OCL-SDK/releases
- For NVidia, see https://developer.nvidia.com/cuda-downloads

With Visual Studio 2015 and an OpenCL SDK installed we can proceed to install PyOpenCL. Open an Anaconda Prompt or a Windows CMD and type:

```
> pip install pyopencl
```

If this completes without errors, PyOpenCL is installed. If you get compilation errors, please set the INCLUDE and LIB environment variables according to your system and try again, e.g. for the CUDA 8 SDK use:

```
> set INCLUDE=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\include
> set LIB=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\lib\x64
> pip install pyopencl
```

The paths listed here assume an NVidia system. Please adapt the paths to your own system and device SDK (e.g. ATI). Important is that the `INCLUDE` path should contain the file `CL\cl.h` and the `LIB` path should contain `OpenCL.lib`. Find these directories if needed. If all goes well, PyOpenCL will be compiled and installed to your system.

If this still does not work, you can try one of the installation guides on https://wiki.tiker.net/PyOpenCL/Installation/Windows, or you can consider (re)installing Anaconda, version >=4.2, with Python 3.5 on your 64-bit Windows system and then try the -cp35-none-win_amd64 wheel linked above.

Installing MOT

With Python and OpenCL installed you can now install MOT. Open an Anaconda Prompt and type:

```
> pip install mot
```

2.4 Testing the installation

Open a Python shell. In Windows you can do this using a the Anaconda Prompt and type `python`. On Linux, use in Bash the `python3` command. In the prompt type:

```
>>> import mot
>>> devices = mot.smart_device_selection()
>>> list(map(str, devices))
```

If you get no errors and the output is a list of CL environments, MOT is successfully installed.

2.5 Upgrading

Ubuntu / Debian Linux

If you used the PPA to install the MOT package, upgrading is easy and is handled automatically by Ubuntu. If you used the pip3 installation procedure you can upgrade MOT with `sudo pip3 install --upgrade MOT`.

Windows

To upgrade MOT when a new version is out, open an Anaconda Prompt or Windows CMD and type:

```
> pip uninstall mot
> pip install mot
```

to upgrade MOT to the latest version.

Chapter 3

Changelog

3.1 Changelog

v0.9.4 (2019-12-06)

Added

- Added the method 'get_subset' to allow getting kernel data for a subset of the instances.

Changed

- Changed the Array kernel data type by removing the offset_str in favor of a simple boolean switch.

Other

- Bugfix in convert_data_to_dtype, in the case a vector type was already in the correct dtype.
- Improved error reporting in the function evaluator.

v0.9.3 (2019-06-04)

Added

- Adds include guards to the Struct kernel data object.

v0.9.2 (2019-04-01)

Added

- Makes CL functions directly callable. Calls are forwarded to the evaluate method.

v0.9.1 (2019-02-21)

Added

- Adds OpenCL approximations to the Gamma CDF and Gamma PPF functions.

Other

- Removed atomic functions. They were unsupport on some platforms.
- Made the ML routines more robust for objective functions that return NaN.

v0.9.0 (2019-02-19)

The primary addition in this version is the support for inequality constraints in the non-linear optimization routines. This allows the user to provide a function $g(x)$ enforcing the rule $g(x) <= 0$. The current optimization routines have no native capability for dealing with these bounds and therefore use the penalty method.

Added

- Adds support for inequality constraints (in the non-linear optimization routines) using the penalty method.
- The CL function parser can now handle `/* */` documentation strings prefixed to the function definition.
- Adds atomic float and double functions to the CL kernels

Changed

- Moved the box constraints to the penalty methods.
- Changed the Richardson extrapolation computation (for the Hessian) to a clearer and cleaner version using recursion.
- In the Levenberg-Marquardt routine, changed the Jacobian computation to a second order approximation. Furthermore, it now uses centered difference by default, and forward and backward difference when near the upper and lower bounds. These changes improve the converge of the LM routine, but do cost more memory.

Other

- The Hessian function is now completely in OpenCL, making it slightly faster.

v0.8.3 (2019-01-08)

Added

- Adds functions for inverting a real symmetric matrix stored as an upper triangular vector.

Changed

- Changed the numerical Hessian to produce upper triangular elements instead of lower triangular.
- Refactored the Hessian function a bit to optionally merge the different functions in the future.

Other

- Renamed solve_cubic_pol_real to real_zeros_cubic_pol.
- Slight refactoring of the Jacobian function of the LM method. The Jacobian is now no longer suspect to the bounds.
- Small refactorings to make the Hessian code more clear.

v0.8.2 (2018-12-11)

Added

- Added routines for computing the inverse of a symmetric matrix using eigenvalue decomposition.

Changed

- Refactored parts of the numerical Hessian routine.

Other

- Fixed LM by adding a double cast.

v0.8.1 (2018-11-23)

Added

- Adds a function for computing the eigenvalues of a 3x3 symmetric matrix.

Changed

- Removed the constants to single precision flag from the compilation.

Other

- Small bugfix in the polynomial computation, no value was returned for 0 real roots.

v0.8.0 (2018-11-09)

This version adds support for box constraints to the optimization routines. For routines that do not natively support box constraints we wrap the evaluation function with a check for bound violations and return INFINITY when that happens.

Added

- Adds support for boundary conditions (box constraints) to the optimization routines.
- Adds composite kernel data private array.

Changed

- Updated the CompositeArray to support multiple address spaces.

Fixed

- Fixed that the LL calculator did not return results for all samples.

v0.7.2 (2018-10-29)

Fixed

- Fixed a small problem when an array with vectors is loaded in the Array class.
- Fixed bug that no device would be selected if only CPU's were present.

Other

- Simplified the parameter objects by moving the data type information into the object. This also makes it simpler to add new types in the future, like the block type in OpenCL 2.0.

v0.7.1 (2018-10-26)

Added

- Adds private memory to the list of KernelData classes. This makes it possible to add private memory elements to the kernel data.

Other

- Small update to the twalk sampler.

v0.7.0 (2018-10-24)

By removing all local variable out of non-kernel functions, this version should now be compatible with POCL (tested with version 1.1).

Changed

- Updates to the samplers. Adds initial t-walk sampler implementation.

Other

- Removed warning filter in the CL function.
- Refactored the constructor of the cl function parameter.
- Removed the CL load balancing. It was not very useful.
- Modified the Nelder-Mead simplex default scales for slightly better fits.

v0.6.14 (2018-10-17)

Added

- Adds linear cubic interpolation method for interpolating on a grid.
- Adds a CL function for Simpson's rule numerical integration

Fixed

- Bugfix in the CL multi-functions parser.

Changed

- Work on moving local variable declarations outside of non-kernel functions. This should in the future allow running MOT on LLVM OpenCL implementations. More work to be done.
- Improved the default settings of the Subplex optimizer.

v0.6.13 (2018-10-08)

Added

- Adds the logpdf for the Gamma and the Normal distribution.
- Adds an example of fitting a Gamma distribution.

Changed

- Changed the function signature of the legendre polynomial computations.

v0.6.12 (2018-10-06)

Fixed

- Out of bounds fix in the legendre polynomial computations.

v0.6.11 (2018-10-04)

Changed

- Updates to the SCAM MCMC routines default settings.

Other

- Removed CL extra from the CL functions and added support for CLCodeObjects for injecting simple CL code as a dependency.
- Small local memory optimization to the LM optimization routine.

v0.6.10 (2018-09-18)

Fixed

- Fixed synchronization bug in the LM optimizer, only happened in very rare cases.

v0.6.9 (2018-09-17)

Fixed

- Fixes the Gamma distribution for nvidia compilers.

v0.6.8 (2018-09-15)

Added

- Adds support for providing a Jacobian in the minimization function.

v0.6.7 (2018-09-12)

Added

- Adds more legendre polynomial functions.

v0.6.6 (2018-09-11)

Added

- Adds CL function to calculate the even Legendre terms of the Legendre polynomial.

v0.6.5 (2018-09-10)

Changed

- Removed (object) declaration from the class declarations, it is no longer needed with Python 3.
- Makes the mot_float_type use typedef instead of macro define.

v0.6.4 (2018-08-28)

- Adds cubic polynomial root finding method for real roots.

v0.6.3 (2018-08-24)

- Another small regression fix.

v0.6.2 (2018-08-24)

- Small regression fix in the Array class, only applicable when using the `offset_str` argument.

v0.6.1 (2018-08-24)

Added

- Adds complex number support using the PyOpenCL complex numbers.

Other

- Small fix to allow scalars in the Array class.

v0.6.0 (2018-08-23)

Changed

- Removed the mot_data_struct system and replaced it with a Struct KernelData object.
- Refactored the numerical hessian API.

v0.5.7 (2018-08-17)

Changed

- Updated lmmin to version 7.0 of http://apps.jcns.fz-juelich.de/doku/sc/lmfit.

Other

- Removed redundant super arguments.
- Simplified the optimization API.

v0.5.6 (2018-08-02)

This version is significantly faster than previous versions when ran using a GPU.

Changed

- Made the optimizers work better with the local reduction.
- Removed some non-ascii characters for compatibility.
- Bugfix to allow using more than one device.

v0.5.5 (2018-08-02)

Changed

- Changed the optimization routines such that they use local memory reduction when evaluating the model. This generally speeds up optimization by 2~5 times.
- Refactored the model interface such that it has the function get_objective_function, instead of objective per observation.
- Restructured the methods to follow more the layout of numpy and scipy.

Other

- Removed get_nmr_parameters and get_nmr_problems from the model interface. This information is already implicit in the starting points.
- Removed the multi-step optimizer and the random restart optimizer.
- Removed NameFunctionTuple, adds parser for CL functions as a string.

v0.5.4 (2018-07-17)

Changed

- Replaced Grako for Tatsu, as Grako was no longer supported.
- Updated makefile to use twine for uploading to PyPi.
- Removed the Tatsu as a debian package and updated installation instructions.
- Removed six as dependency.

v0.5.3 (2018-07-16)

Changed

- Small enhancements to the function evaluator.

v0.5.2 (2018-07-05)

Fixed

- Fixed model proposal updating using the model. The parameter vector was not correctly reset if more than one parameter was updated.

v0.5.1 (2018-07-01)

Added

- Adds support for nested structures in the kernel input data.

v0.5.0 (2018-06-01)

This version removes support for Python version <= 2.7. Now only Python > 3 is supported.

Added

- Adds Ubuntu 18.04 release target.

Changed

- Removes Python version <= 2.7 support.

Other

- Removed the gaussian/mean/median filters from the package. If this is needed in the future it would be better to support it as list-processing kernels instead of 3d volume filters.

v0.4.4 (2018-05-15)

Added

- Adds the PDF, CDF and PPF (Quantile function) of the Normal and Gamma distribution as reusable CL functions.

v0.4.3 (2018-05-03)

Added

- Adds gamma pdf CL function.

Changed

- Improved the runtime efficiency of the ProcedureRunner by allowing a workgroup size of None.
- Renamed get_nmr_inst_per_problem to get_nmr_observations.
- Updated to the function evaluate signature to use the cl_runtime_info object.

Other

- Refactored the optimization routines to use the RunProcedure paradigm.
- Made the compile flags a list instead of a dict.

v0.4.2 (2018-04-11)

Added

- Adds some modeling examples.

v0.4.1 (2018-04-09)

Added

- Adds random scan to the Random Walk Metropolis algorithms.

Other

- Renamed 'get_nmr_estimable_parameters' to 'get_nmr_parameters'
- Moved the model building modules to MDT.
- Removed the eval function from the model interface.

v0.4 (2018-04-04)

This release provides a cleaner interface to the optimization and sampling routines. Furthermore, it improved the decoupling between the models and the MCMC samplers allowing to, in the future, add more MCMC samplers.

Added

- Adds additional patience parameter for the line search in the Powell algorithm.

Changed

- Completely restructured the MCMC sampling routines by decoupling the proposal distributions from the model functions.
- Removed some weight models from the model builder and moved those to MDT.
- Removed the 'get_initial_data' method from the model interface.

Other

- Renamed dependency_list to dependencies in the models and library functions.
- Renamed parameter_list to parameters in the model functions.
- Small caching and object initialization updates.

v0.3.12 (2018-02-22)

Added

- Adds CL context cache to fix issue #5.
- Adds singularity boolean matrix to the output of the Hessian to covariance matrix.

v0.3.11 (2018-02-16)

- Simplified the CL context generation in the hope it fixes issue #5.

v0.3.10 (2018-02-14)

Changed

- Changed the default load balancing batch size.

v0.3.9 (2018-01-30)

Added

- Numerical Hessian now with OpenCL support
- Adds method to get the initial parameters of a model.
- Adds initial lower and upper bound support to the numerical Hessian method.
- Adds a method to the sampling statistics to compute the distance to the mean.
- Adds InputDataParameter as superclass of ProtocolParameter and StaticMapParameter.
- Adds support for restrict keyword in CL functions.

Changed

- Updates to the numerical Hessian calculation, translated more functions to OpenCL.
- Updated the buffer allocation in some methods to the new way of doing it.
- Updates to the numerical Hessian calculation, small improvement in local workgroup reductions.
- Changed the interface of the input data object to get the value for a parameter using a method call.

Other

- Sets the default step size to 0.1 for the numerical differentiation, small updates to the numerical Hessian computation.
- Most of the numerical Hessian computations are now in OpenCL. Only thing remaining is median outlier removal.
- Made the KernelInputDataManager smarter such that it can detect duplicate buffers and only load those once. Furthermore, KernelInputScalars are now inlined in the kernel call.
- Made the method wrapping in the wrapped model easier.
- Lets the random restart use the model objective function instead of the L2 error. Furthermore, removed residual calculations in favor of objective function calculating.
- Renamed EvaluationModels to LikelihoodFunctions, which covers the usage better.
- Removed the GPU accelerated truncated gaussian fit since it was not doing the right thing. Added a MLE based truncated normal statistic calculator.
- In MCMC, changed the order of processing such that the starting point is stored as the first sample.

v0.3.8 (2017-09-26)

- Small fix to the work group size, this will fix a INVALID_WORK_GROUP_SIZE issue with the procedure runner.

v0.3.7 (2017-09-22)

Added

- Adds a GPU based truncated gaussian fit.
- Adds a GPU based univariate ESS algorithm.

Changed

- Updates to the model function priors.
- Updates to the KernelInputDataManager.
- Changed the sample statistic to use the CPU again for the easy statistics, for large samples this is faster than using the GPU.
- Updates to the function evaluator, made the input argument r/w by default and allows for void output functions.

Other

- Prepared new release.
- Refactored the residual calculator, small performance update in MCMC.
- Removed two old mapping routines, the objective calculators.

- Project renaming.
- Work on the log likelihood calculator.
- Simplified some sampling post processing after changes in MOT.
- Removed the GPU multivariate ESS again, it was only marginally faster.
- Small speed update to the GPU univariate ESS method.
- More work on the procedure evaluator. Moved more data management tasks to the kernel input data manager.
- Renamed CLHeader to CLPrototype, covers the usage better.

v0.3.6 (2017-09-06)

Added

- Adds CL header containing the signature of a CL function. Modified the evaluation models to not be a model but contain a model.
- Adds a method finalize_optimized_parameters to the optimize model interface. This should be called once by the optimization routine after optimization to finalize the optimization. This saves the end user from having to to this manually in the case of codec decorated models.
- Adds mot_data_struct as a basic type for communicating data to the user provided functions.

Fixed

- Fixed the rician MLE estimator. The square root was missing since the optimization routines do the squaring.

Other

- Converted all priors to CLFunctions.
- Instead of the square root in the model, we take the square root in the LM method instead.
- Made the KernelInputData not contain the name, but let the encapsulating dictionary contain it instead. Made more things a CLFunction and made the library functions such that the contain just one function (trying to). Updates to the evaluation model to be more of a builder for the LL and evaluation function rather then to have the evaluation model be a function itself. The latter needs more work.
- Aligned the interface of the NamedCLFunction with the CLFunction for a possible merge in the future.
- Refactored the interface of the CLFunction class from properties to get methods.
- Small updates in various places. Local memory bug fix in the sampler.
- Made two functions for the Gamma functions.
- Made the library and model functions a subclass of a CLFunction. Adds a general CL procedure runner and a more specific CLFunction evaluator to the mapping routines. Adds the method `evaluate` to the CLFunction class such thatit is possible to ask a model to evaluate itself against some input."
- Moved the mot_data_struct generation from the model to the kernel functions.
- More changes to adding the mot_data_struct type.
- Intermediate work on the sampling mle and map calculator.

v0.3.5 (2017-08-29)

Added

- Adds support for static maps per compartment overriding the static maps only per parameter.

Changed

- Updated the changelog generation slightly.
- Updated the problem data to be a perfect interface.
- Updates the parser to the latest version of Grako.

Fixed

- Fixed the link to the AMD site in the docs.

Other

- Renamed AbstractInputData to just InputData, which is more in line with the rest of the naming scheme.
- Renamed problem data to input data.
- Code cleanup in and variable renaming.
- Removed get_free_param_names as a required function of a model.
- Removed the DataAdapter and in return added a util function convert_data_to_dtype.

v0.3.4 (2017-08-22)

Added

- Adds a residual CL function to the model.

Other

- Removed the const keyword from the data pointer in the model functions. Allows the user more freedom.
- Removed the get observation return function from the model interface.

v0.3.3 (2017-08-17)

Added

- Adds gitchangelog support for automatic changelog generation.
- Adds a positive constraint to the library.
- Adds the get_composite_model_function() function to the model builder returning a barebones CL version of the constructed model.

Changed

- Updates to the changelog.

Fixed

- Fixed WAIC memory.
- Fixed small indexing problem in the sampler.

Other

- Small updates to the interfaces. Different batch size mechanism in MH, works now with larger number of parameters.
- Removed support for dependencies in the parameter transformations.
- Moved the cartesian product method to the utils.
- Small fix in handling static maps.
- Makes sure the calculated residuals are always a number and not NaN or INF.
- Small cosmetic changes.
- Small updates to the documentation. CLFunctionParameter now accepts strings as data type and will do the conversion itself.

v0.3.2 (2017-07-26)

Changed

- Update to the documentation

v0.3.1 (2017-07-25)

Added

- Adds some Gamma functions with tests.

Other

- The model builder now actually follows the builder pattern, as such model.build() needs to be called before the model and the result needs to be passed to the optimization routines.
- Adds Gamma related library functions
- Removed the objective function and LL function and replaced it with objective_per_observation and LL_per_observation.
- Introduced get_pre_eval_parameter_modifier in the model interface for obvious speed gains.
- Undid previous commit, it was not needed.
- Small update to allow the model to signal for bounds.
- Some updates to work with static maps in the model simulation function.
- Small update to the calculation of the dependent weight (the non-optimized weight could have been smaller than 0, which is not possible).
- Made the processing strategy log statement debug level instead of info level.
- Refactored the model builders to the actual builder pattern. Small change in the OffsetGaussian objective per observation function to properly account for the noise. Removed the objective function and LL function and replaced it with objective_per_observation and LL_per_observation. Introduced get_pre_eval_parameter_modifier in the model interface for obvious speed gains.
- Introduced the KernelDataInfo as an intermediate object containing the information about the kernel data of the model.

v0.3.0 (2017-06-11)

Added

- Adds fixed check in the init value method. This to prevent overwriting fixations by initialization.
- Added priors to the model functions.
- Add a routine that calculates the WAIC information criteria.

Changed

- Changed support for the post optimization modifiers. Small change in the sampling statistics.
- Changed the rand123 library such that it no longer automatically adds the global id to the random state. Initializing the proper state is now part of the caller.

Fixed

- Fixed small regression in the model builder, it did not correctly read in the fixed values.

Other

- The get_extra_results_maps function of the compartments now receives and gives the dictionaries without the compartment name, making things easier.
- Moved the data from the model builder to the ModelFunctionsInfo class.
- Adds a mechanism for adding model wide priors.
- Removed redundant comment Refactored one of the priors.
- Moved the codec out of the optimization routines.

Chapter 4

Credits

The Multi-threaded Optimization Toolbox is a model optimization toolbox originally meant for diffusion MRI analysis, but applicable to other fields as well. Software development by Robbert Harms, under the (Phd) supervision of Alard Roebroeck, at Maastricht University.

4.1 Contributors

- **Robbert Harms**
  - Lead developer