InsightToolkit Kinetic Analysis (itk::ka) Library

Release 1.00

Nicholas Dowson[1], Charles Baker[1,3], David Raffelt[4], Jye Smith[2], Paul Thomas[2,3], Olivier Salvado[1] and Stephen Rose[1,3]

September 23, 2014

[1]Australian eHealth research Centre, Level 5, UQ Health Sciences Building, Royal Brisbance and Women's Hospital, Herston, QLD, 4029, Australia
[2]Specialised PET Services, Queensland, Royal Brisbance and Women's Hospital
[3]Faculty of Medicine and Biomedical Science, University of Queensland
[4]The Florey Institute of Neuroscience and Mental Health, Melbourne

Abstract

This paper describes how to compile and use the Insight Toolkit Kinetic Analysis library (itk::ka), to perform analysis of dynamic medical images, along with a brief overview of the library source code. A set of interfaces to facilitate application development (koala) and an application using Powell's Dogleg optimisation (koalaDogleg) are also supplied, along with some sample data. Only versions of ITK4 and higher are currently supported.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3469]

Contents

1 Introduction

Kinetic analysis is useful for examining dynamic PET and MR images, however many existing kinetic analysis applications are costly, supplied in binary format only or are targeted at a specific platform. Also, although many of the individual components for kinetic analysis applications are available, assembling these into a single application, and the selection and tuning of the internal algorithms is time consuming.

However, the pipeline architecture, extensive image IO support, large user community, and relatively seamless multi-threading provided by the Insight Toolkit made it an excellent springboard for the purposes of a kinetic analysis library. To the authors' knowledge, kinetic analysis has yet to be added as a capability of the InsightToolkit, hence the development of the itk::ka library.

Additional, necessary capabilities are provided by the SUNDIALS library for solving ordinary differential equations, libxml2 for accessing xml parameter files, and various C++ Boost libraries.

This document serves to briefly describe how to compile and use the itk::ka library, the facades (koala) and kinetic analysis application (koalaDogleg) provided. The source code is distributed under a BSD-style license.

2 InsightToolkit Kinetic Analysis Class Overview

The code framework is based on several groups of classes to encode key concepts, which are discussed in this section.

Three namespaces were used, the baseline `itk` namespace for classes that are not specific to kinetic analysis, such as optimisation or reading and writing vector images, the `itk::ka` namespace for classes specific to kinetic analysis and the `koala` namespace for source code specific to interfacing to the commandline or Matlab^{TM}.

This section only provides an overview of the library source. For more detail the doxygen documentation should be consulted.

2.1 Time Activity Curves

The most primitive data type in the class is the `itk::ka::Tac` class, which defines a time activity curve. Inheriting from this is the `itk::ka::DataTac` class, which adds the ability to scale and shift time activities reversibly albeit with interpolation artifacts. The class `itk::ka::Bif` defines a blood input function and further inherits from `itk::ka::DataTac` by including a container for the duration of each frame and some blood input function specific methods.

The class `itk::ka::SimulatedTac` inherits from `itk::ka::Tac` and defines any time activity curve that is generated according to some parameters given a particular model. TACs which are being optimised are defined using the `itk::ka::OptimisedTac` class, which inherits from `itk::ka::DataTac`, but also includes a member for the best matching simulated TAC so far, and the associated match score. The relationship between the TAC classes is shown in Fig. 1.

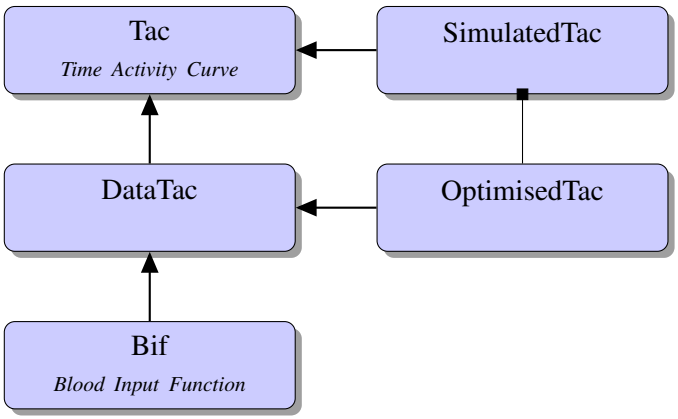

Figure 1: Time Activity Curve Classes

To perform kinetic analysis using the pipeline model of itk, time activity curves are grouped together into a container called `itk::ka::TacSet` which inherits from `itk::ImageBase<3>`, as shown in Fig. 2. This makes the implicit assumption that TACs are stored in 3D arrays of TACs in an image within a particular spatial domain, *i.e.* an image sourced from a PET scanner with an origin, spacing and orientation. This class could be generalised to N-Dimensional images, but the extra code boilerplate overhead is not currently justified. Note that an actual image grid is not stored by the `itk::ImageBase<3>` class, only the information on the image grid which is passes along the processing pipeline. This was very useful for the `itk::ka::TacSet` class, which also stores the original time activity curves and the model being used to fit

the TACs.

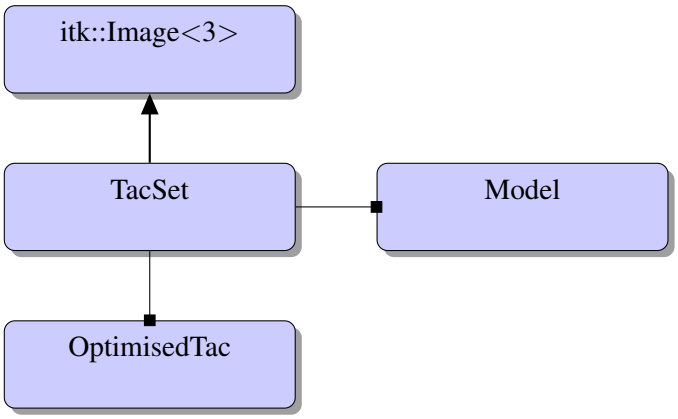

Figure 2: Time Activity Curve Set Class

2.2 Filters and optimisation

To actually generate and process the `itk::ka::TacSet` objects, are several new filters. `itk::ka::TacSetFromImageFilter` converts a (3+1D) vector image to a `itk::ka::TacSet`, *vice-versa* for `itk::ka::TacSetToImageFilter`. In addition, the `itk::ka::TacSetOptimiser` filter optimises a particular set of TACs, and the `itk::ka::TacSetFromParamFilter` generates a `itk::ka::TacSet` from a vector image of parameters.

The supplied `itk::VectorImageFilterFileReader` and `itk::VectorImageFilterFileWriter` allow the reading and writing of image files for vector images.

The algorithm the optimiser filters use are specified by supplying a particular optimisation algorithm. Several of these exist, as shown in Fig. 3 including:

itk::ka::OptimiserInitial to initialise TACs to one starting point

itk::ka::OptimiserBruteForce to the perform a brute search through a set of predefined simulated TACs defined on a parameter grid

itk::ka::OptimiserTwoStepInitialiser to brute force initialise for a grid of *non-linear* parameters while performing a non-negative least squares optimisation of the remaining *linear* parameters.

itk::ka::Optimiser[One,Two,Irreversible CompartmentTwoStepInitialiser] specialise the two step initialiser to a particular compartment model.

itk::ka::OptimiserPowellDogleg optimises using the Powell dogleg algorithm.

The remaining optimisers are stubs that act as ancestors for new optimiser types.

Together the `itk::PowellDogleg`, `itk::QuasiNewtonNonlinearOptimiser` and `itk::QuasiNewtonCostFunction` define the Powell Dogleg algorithm[4], as shown in Fig. 4. These classes were necessary to generalise the manner in which the residual and jacobian were calculated. The `itk::ka::NonNegativeLeastSquaresSolver` class can minimise linear problems while imposing non-negative constraints on the parameters using the Lawson's algorithm [3] with some modifications suggested by Bro and de Jong [1].

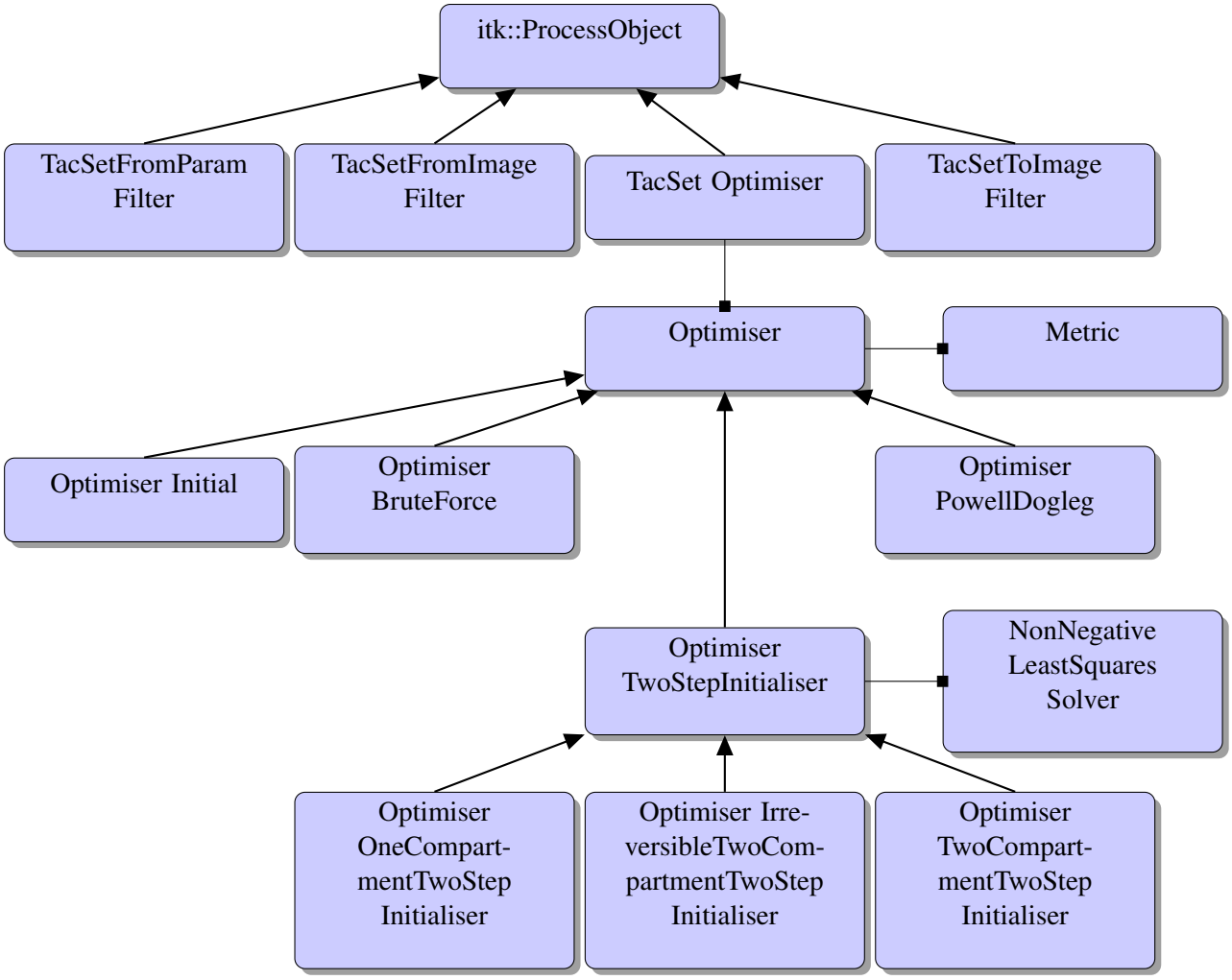

Figure 3: Kinetic Analysis Filters and Optimisers

2.3 Metric

Kinetic analysis is performed by minimising a given TAC according to a given metric. Currently, just one metric is supplied: `itk::ka::Metric`, which can measures L_2 distance with some optional parameter regularisation:

$$D = \sum_i [T(t_i) - S(t_i;\mathbf{k})]^2 + \lambda \sum_j |\mathbf{k}_j|^\alpha \quad (1)$$

where λ and α control the amount of regularisation applied. In the future this will be generalised to measure the L_1 distance as well.

2.4 Models and Simulators

The minimisation is performed in the context of a particular model, which defines the set of all time activity curves that can be generated for a given blood input function, a set of frame times and given parameterisation. The base class `itk::ka::Model` is *not* abstract because it is used by the `itk::ka::TacSet` class to disseminate key information along the processing pipeline. Inheriting from this class are specific parame-

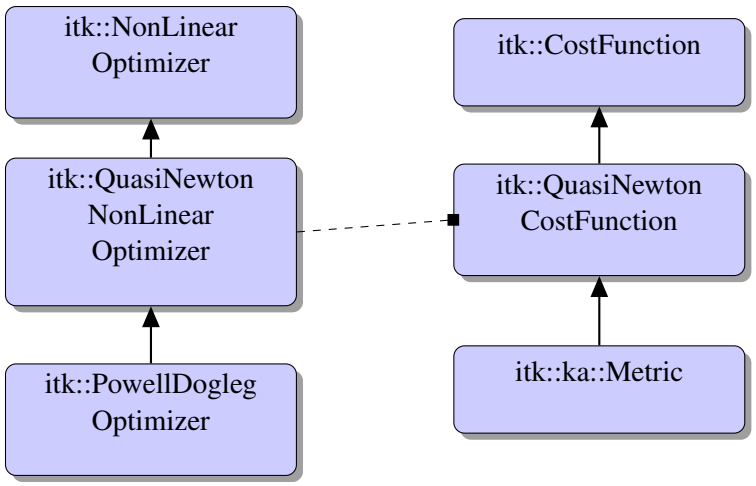

Figure 4: Classes used for Powell Dogleg Optimisation

terisations called simulators. Currently the supplied simulators are all ODE-based formulations of compartmental models based on the SUNDIALS toolkit. An interface to SUNDIALS [2] is supplied using the facade pattern by `itk::ka::SimulatorSundialFacade`, from which the one, irreversible two, and two compartment models inherit respectively defined by `itk::ka::SimulatorStd1c`, `itk::ka::SimulatorStd2ic`, and `itk::ka::SimulatorStd2c` classes, as shown in Fig. 5.

Other (non-ODE-based) formulations are expected to be released in the future. In addition a simulator factory class is supplied, `itk::ka::SimulatorFactory`, which can generate models based on a name string.

2.5 Other classes

A commandline interface to the kinetic optimisation library in `koalaDogleg.cxx` file, with application global variables defined in `koalaGlobals.*`, useful utilities in `koalaUtilities.*`. Commandline arguments and xml parameters are processed by the `koalaArgumentManager` class.

For convenience a pipeline of optimisers is specified in koala using the `itk::ka::KineticAnalysisPipeline`, which contains a full processing pipeline as illustrated in Fig. 6.

Some interfaces to Matlab are defined in the `matlab` subfolder.

3 Software Requirements

The koalaDogleg application and itkka library depend on several other libraries:

SUNDIALS for the ODE solver, is available here: `http://computation.llnl.gov/casc/sundials/main.html`

The Insight Toolkit with powerful pipelining, multithreading and medical image IO/manipulation routines, and large helpful user community. `http://www.itk.org/ITK/resources/software.html`. Note: only versions of ITK 4 or higher are supported.

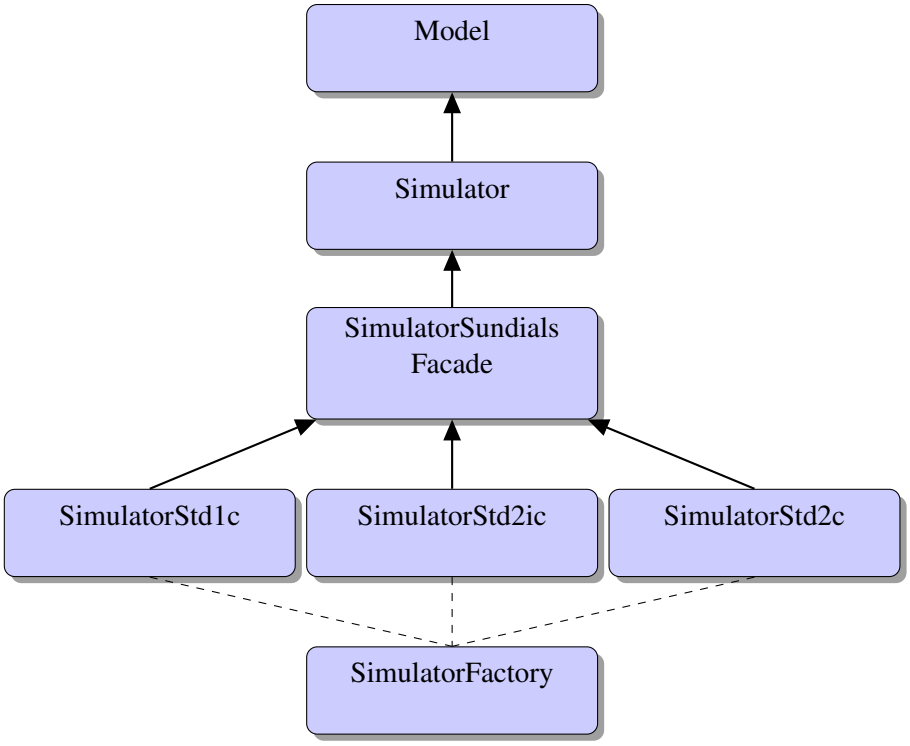

Figure 5: Simulator Classes, which can be created by the factory

libxml2 for reading and writing the xml files used to control koalaDogleg `http://xmlsoft.org/downloads.html`

Boost C++ for several purposes including program options, shared ptr, timer, array, serialization, property tree, timers, etc. `http://www.boost.org/users/download/`

These libraries were selected both because they are well maintained and have good interfaces and because they have BSD/MIT style licenses.

All of the libraries will need to be downloaded and installed. In each case, their source can be downloaded from the URLs listed above and compiled based on their respective instructions.

However it is probably more convenient to use your operating system's package manager for this purpose. Instructions for Ubuntu are supplied in the next subsection.

3.1 Instructions to install dependencies on Ubuntu 14.04

1. First add the neurodebian repository to your package manager so you can conveniently get hold of the latest ITK, as described at `http://neuro.debian.net/`:

```
wget -O- http://neuro.debian.net/lists/trusty.au.full | \
  sudo tee /etc/apt/sources.list.d/neurodebian.sources.list
sudo apt-key adv --recv-keys --keyserver hkp://pgp.mit.edu:80 2649A5A9
sudo apt-get update
```

There is often an error message, which does not appear to have a discernable effect: "tee: unrecognised option '–recv-keys'"

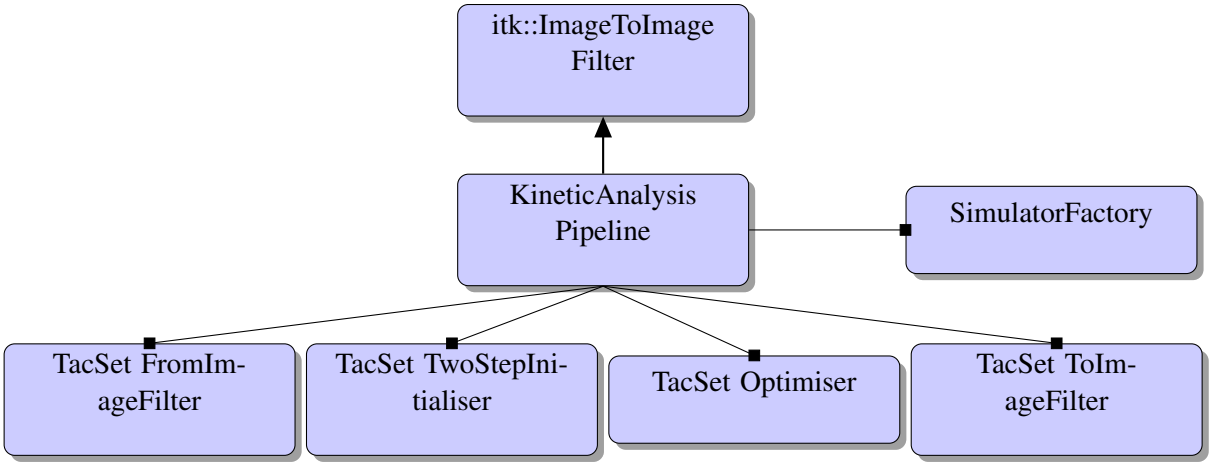

Figure 6: Members of the kinetic analysis pipeline class

2. Install the cmake build system and its gui

```
sudo apt-get install cmake cmake-curses-gui
```

3. Install ITK

```
sudo apt-get install libinsighttoolkit4-dev
```

4. Install all the boost libraries

```
sudo apt-get install libboost-all-dev
```

5. Install SUNDIALS

Web browse to:

http://computation.llnl.gov/casc/sundials/download/download.html

Click on sundials-2.5.0.tar.gz

Run the following:

```
tar zxf sundials-2.5.0.tar.gz
cd sundials-2.5.0
mkdir build
cd build
cmake ..  -DBUILD_STATIC_LIBS=FALSE -DBUILD_SHARED_LIBS=TRUE -DCMAKE_BUILD_TYPE=Release
make
sudo make install
cd ../..
sudo ldconfig
```

3.2 Compilation of koala

To compile the koala application do the following.

```
mkdir build
cmake ..  -DCMAKE_BUILD_TYPE=Release
make
```

4 Usage for koala

Once koalaDogleg is compiled, you can browse to the test folder and run it on the test data. Note that you will need to have downloaded the test data file and untarred it as well (i.e. `tar zxf itkka-data-140909.tgz`):

```
cd ../test
../build/koalaDogleg --xml test_param_std2ic.xml
```

You can view the options that can be configured in the xml file by typing:

```
cat test_param_std2ic.xml
```

Note that the file with the input PET data is specified in the xml file, and that the filepath specified is relative to the current working directory, NOT the xml file.

Commandline options are also provided, but the large number of inputs required makes this less practical and it makes it difficult to document experiments. hence command line inputs are being partly phased out, and are not documented here. However, typing `koalaDogleg --help` will list available options.

4.1 Description of parameters

The example xml file is used to describe each of the parameters for koalaDogleg is shown in Table 1. Many of the options are defined in the `koalaArgumentmanager.h` source file, or at least this file can be used to see where the relevant code lies.

Table 1: Outline of xml file used to configure koala.

```
<!--Header-->
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!--Group koala parameters-->
<Koala>
  <!--Maximum no. threads-->
  <Threads> 24 </Threads>
  <!--Control text output. Range is 0 to 4.-->
  <Verbosity> 1 </Verbosity>
  <!--Input units. Can be cnts or cps.-->
  <InUnits> cnts </InUnits>
  <!--Output units. Can be cnts or cps.-->
  <OutUnits> cnts </OutUnits>
  <!--Mask out data where ∑time is lower. ANDed with mask image (if any).-->
  <MaskingThreshold> 150000 </MaskingThreshold>
  <!--Weight frames by: data, duration, uniform. -->
  <WeightMode> data </WeightMode>
  <!--Which kinetic model: Std1c, Std2ic, Std2c. See itkkaSimulatorFactory.h for options.-->
  <Model> Std2ic </Model>

  <!--No. frames to seek TAC spike to align with BIF.-->
  <SpikeWindowAtOptimisation> 10 </SpikeWindowAtOptimisation>
  <!--Initialation strategy: single-point or brute-search. See itkkaOptimiserInitial.h and itkka*Initialiser.h-->
  <InitialisationMethod> brute-search </InitialisationMethod>
  <!--Image being analysed-->
  <InDynamicPetImage> testdata.nii.gz </InDynamicPetImage>
  <!--Input mask image (if any). ANDed with thresholded image.-->
  <InMaskImage> </InMaskImage>
  <!--Input parameters to initialise data.-->
  <InParametersImage> </InParametersImage>

  <!--Output parameters image-->
  <OutParametersImage> test_out_param_std2ic.nii.gz </OutParametersImage>
  <!--Output simulations image-->
  <OutSimulationImage> test_out_sim_std2ic.nii.gz </OutSimulationImage>
  <!--Output Error image.-->
  <OutErrorImage> </OutErrorImage>
  <!--Output of ANDed input mask and threshold mask.-->
  <OutMaskImage> </OutMaskImage>
  <!--Output of simulation split into individual compartments.-->
  <OutFullSimulationImage> test_out_fullsim_std2ic.nii.gz </OutFullSimulationImage>

  <!--Define time frames for BIF. Must have same no. entries as BloodInputFunctionValuesInCps.-->
  <BloodInputFunctionTimes> 0.1667, 0.3333, 0.5, 0.6667, 0.8333, 1, 1.5, 2, 2.5, 3, 4, 5, 10, 15, 20, 30, 40, 50, 60 </BloodInputFunctionTimes>
  <!--Define BIF values for linearly interpolated BIF. Must have same no. entries as BloodInputFunctionTimes.-->
  <BloodInputFunctionValuesInCps> 0, 120, 424, 101, 178, 86, 86, 78, 73, 68, 64, 59, 51, 43, 39, 35, 32, 29, 27 </BloodInputFunctionValuesInCps>
  <!--Length of each PET frame. Independent of BloodInputFunctionTimes/Values.-->
  <FrameDurations> 0.167, 0.167, 0.167, 0.167, 0.167, 0.167, 0.5, 0.5, 0.5, 0.5, 1, 1, 5, 5, 5, 10, 10, 10, 10 </FrameDurations>
<!--End of koala parameters-->
</Koala>
```

5 Results

The code has been tested on numerous PET scans in multiple anatomical regions. Multithreading allows the 9078 tacs in the test data to be processed in approximately 30 seconds (690 thread.seconds) on a 24-core i7. The central slice of the input data is shown in Fig. 7 along with the time activity curves at the four indicated positions. The fitted activities in each compartment of the model along with fitted time activity curves are shown in Fig. 8.

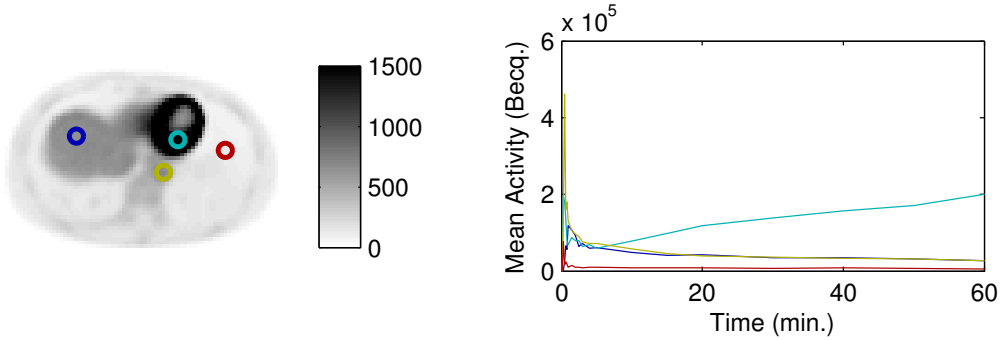

Figure 7: The input data and the time activity curves of four selected points.

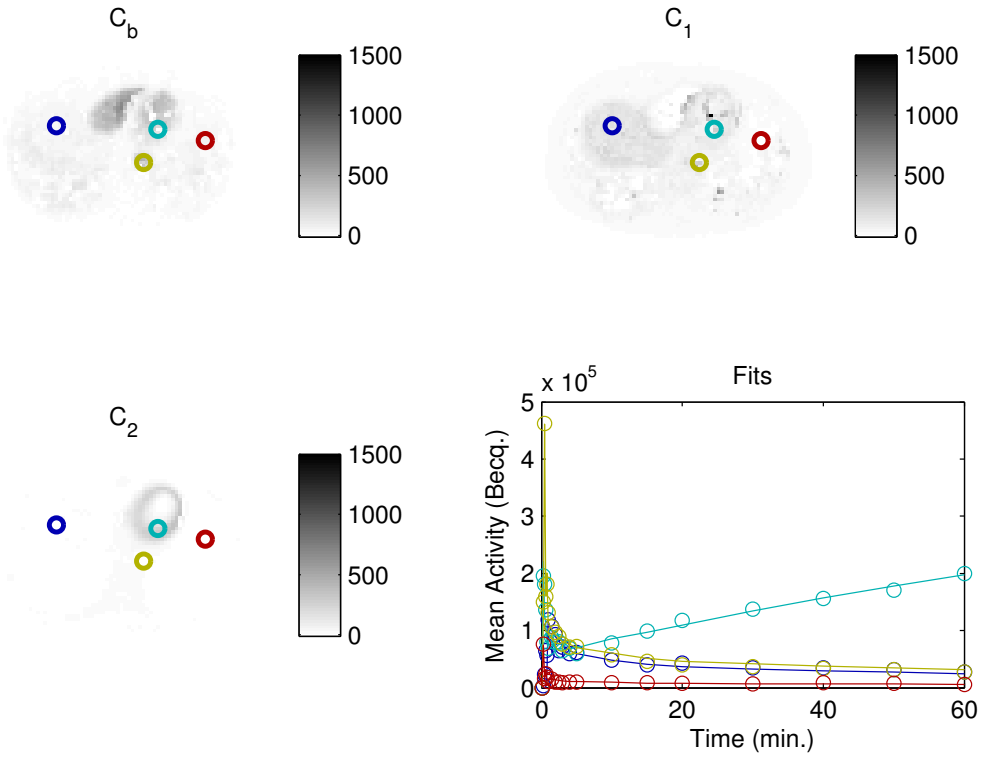

Figure 8: The output fits showing the three compartments, and the fitted time activity curves at each selected point.

6 Test Data

To files are contained by the test archive (`itkka-data-140909.tgz`):

testdata.nii.gz A few slices from an FDG scan of a liver.

test_param_std2ic.xml A file storing the input parameters to perform the kinetic analysis on the test data.

7 Acknowledgements

The authors would like to acknowledge the financial support of the Queensland Cancer Council, the Australian National Health and Medical Research Council (grant number: 631567), and CSIRO.

The authors would also like to thank Jason Dowling and David Rivest-Henault for testing and providing many helpful comments.

Thank you also to the many developers and supporters of ITK, SUNDIALS, libxml2 and the C++ boost libraries.

References

[1] R. Bro and S. de Jong. A fast non-negativity-constrained least squares algorithm. *Journal of chemometrics*, 11:393–401, 1997. 2.2

[2] Alan C. Hindmarsh, Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward. SUNDIALS: suite of nonlinear and Differential/Algebraic equation solvers. *ACM Trans. Math. Softw.*, 31(3):363396, September 2005. 2.4

[3] Charles L Lawson, Richard J Hanson, and Society for Industrial and Applied Mathematics. *Solving least squares problems "This SIAM edition is an unabridged, revised republication of the work first published by Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974"–T.p. verso*. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), Philadelphia, Pa., 1995. 2.2

[4] E. Mizutani. Powell's dogleg trust-region steps with the quasi-newton augmented hessian for neural nonlinear least-squares learning. In *International Joint Conference on Neural Networks, 1999. IJCNN '99*, volume 2, pages 1239 –1244 vol.2, July 1999. 2.2