Introduction to LTPDAworkbench

M Hewitson 22-11-08

The LTPDAworkbench class is a MATLAB class which wraps a set of java classes. They present the functionality of a visual programming interface where blocks representing different LTPDA algorithms can be joined together with pipes to represent a data analysis pipeline.

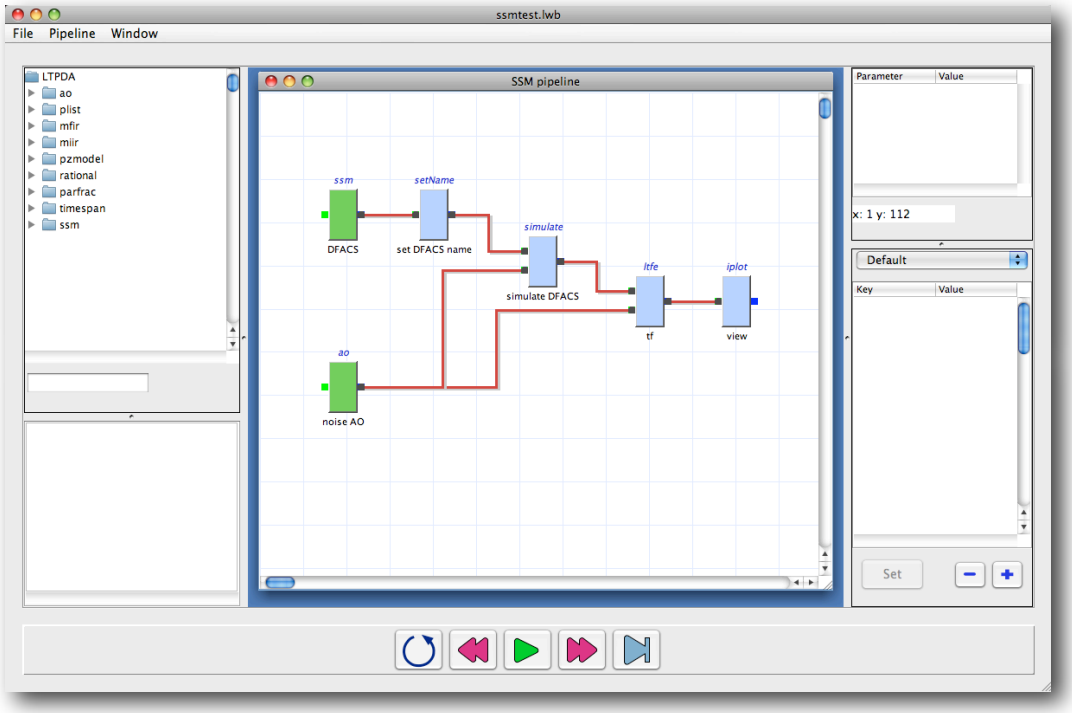
Concepts and terminology

An LTPDAworkbench consists of
1) A library of LTPDA algorithms
2) A panel to set block properties
3) A panel to set algorithm parameters
4) A central container for multiple pipelines

Each pipeline in the workbench is a collection of blocks connected by pipes. Each block represents a particular LTPDA algorithm. A block can have multiple input and output ports. An output port can be connected to multiple pipes, but an input port can only be connected to one pipe. All blocks on a given pipeline are identified by their names, which must therefore be unique. The LTPDAworkbench handles the block names in most cases. If a block with a name of that of an existing block is added to the pipeline, it is renamed by appending '_1'.

Blocks hold an active plist, as well as the full sets of plists given by the minfo object that represents that LTPDA algorithm. Currently, all parameters are entered and presented to the user as strings. We have to see if this turns out to be a limitation.

The following figure shows a new LTPDAworkbench with no open pipelines.



Creating a new empty workbench

To create a new workbench in MATLAB, execute the following command:

>> wb = LTPDAworkbench;

The variable ‘wb’ is now a handle pointing to the instance of the LTPDAworkbench class.

The result should be an empty workbench with no pipeline documents.

On the top left of the workbench you have the library of available LTPDA algorithms.
On the bottom left is a panel that displays the help text of any selected algorithm.
On the top right we have a parameter panel for block parameters.
On the bottom right we have a parameter panel for the algorithm parameters.

Saving a workbench

LTPDAworkbench objects are saved to disk in an XML format with a file extension ‘.lwb’.

Saving a workbench can be achieved in three ways:

1) cmd-s on the workbench will save the workbench with it’s current name. If the workbench is ‘Untitled’, the user will be prompted for a file name.
2) On the workbench, select File->Save
3) From the MATLAB prompt:
    1) >> wb.save(‘foo.lwb’)

Loading a workbench

To load a workbench from disk,

>> newwb = LTPDAworkbench(‘foo.lwb’)

Combining workbenches

If you have an open workbench and you want to add the pipelines from another workbench which is saved on disk, you can do ‘cmd-o’ on the open workbench, or File->Open, and select the appropriate file. This should add the pipeline documents from the workbench on disk to the open workbench.

Create a new pipeline in an existing workbench

To add a new pipeline to the current workbench, you can do
1) cmd-n (or File->New) on the workbench, or
2) File->New, or
3) on the MATLAB prompt:

1) >> wb.newPipeline(); % unnamed pipeline
2) >> wb.newPipeline('My pipeline'); % named pipeline

You can also create a new pipelines from existing LTPDA user objects:

>> wb.newPipeline(objs);

Graphically programming a pipeline

To add blocks to the currently selected pipeline, locate the block in the library tree, right-click on it, and select ‘add block’.

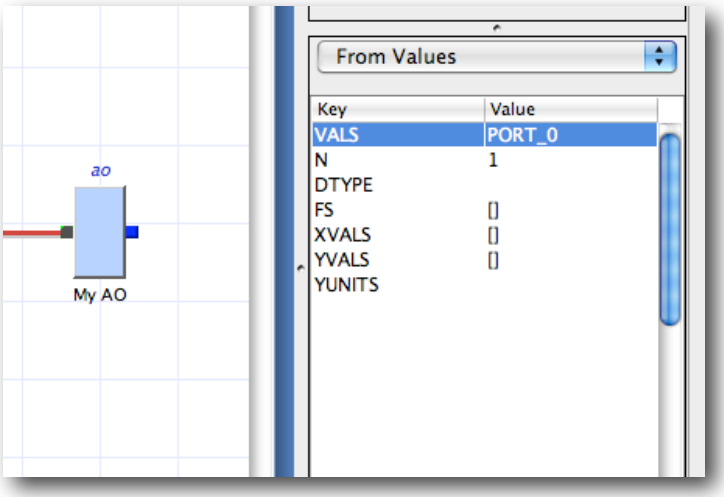
To add inputs or outputs to a block, right-click on the block and select ‘add input’ or ‘add output’ accordingly.

To rename the pipeline, right-click on the pipeline canvas and select ‘Set Pipeline Name’. You can also do Pipeline->Rename.

To set the properties of a block, click on a block to select it. The properties panel in the top right corner of the workbench then shows any settable properties. To set a new property value, double-click the value in the table. Press <enter> to accept the new value.

To set algorithm parameters of a block, click a block to select it. The parameters table in the bottom right of the workbench now shows the current parameter list for that block. To choose from one of the built-in parameter sets, select that set in the drop-down menu, edit any parameter values (or keys) by double-clicking on the table, then click the ‘set’ button to set that plist to the block. You can also add/remove parameters to/from the list by clicking the ‘+’ and ‘-’ buttons.

If you want a parameter value to come from the output of another block, use a value of ‘PORT_#’ where ‘#’ is replaced by the port number** where the source block is connected. For example:



*** currently the numbering of ports on the MATLAB side follows MATLAB convention, 1->N. On the java interface, it follows java convention, 0->N-1. This can easily be made uniform later.*

Currently supported features of the java interface:

1) multiple pipelines
2) copy-and-paste on and between pipelines and workbenches.
3) Select and move blocks.
4) scroll the pipeline canvas with <alt>-mouse drag
5) select all blocks with <cmd>-a
6) save workbench with <cmd>-s

Execution controls

Each block of a pipeline has three states: idle (blue), executed (orange), ready to execute (green).

Running a pipeline is achieved using the controls at the bottom of the workbench.

Reset
Step Backwards
Run
Step Forwards
Skip Forwards

Reset: resets the current pipeline so that the first executable blocks are ready to go.
Step Backwards: Move the block cursor back one step
Run: reset and step through all blocks executing as we go
Step Forwards: execute the current 'ready' (green) blocks and move forward one step
Skip Forwards: move forward one step without executing the current ready blocks.

Creating a pipeline programatically

You can construct a pipeline from within MATLAB by executing commands. The following commands are currently supported:

addBlock

adds a block to the current pipeline. The input arguments are:

addBlock(name, minfo, x, y)

name - a name for the block.
minfo - an minfo object describing the algorithm that this block presents
x - the x co-ordinate of the block on the canvas
y - the y co-ordinate of the block on the canvas

For example:

>> wb.addBlock('My AO', ao.getInfo, 10,10)

adds an AO constructor at position (10,10) to the current pipeline.

connectBlocks

connects two blocks at the given ports. Ports are numbered 1 to N. The arguments are

connectBlocks(sourcBlock, outport, destBlock, inport)

sourceBlock - the name of the source block for the connection
outport - the number of the output port to connect from
destBlock - the name of the destination block
inport - the number of the input port to connect to

For example,

>> wb.connectBlocks('My AO', 1, 'My LPSD', 3)

If the request port doesn't exist, all ports up to that number are created. So, for example, in the previous command, if the 'My LPSD' block had only one input, then this command would cause two additional inputs to be created.

uploadPlist

uploads a plist object to a block. The input arguments are:

uploadPlist(blockName, pl)

blockName - the name of the block to upload to
pl - a plist to upload

For example,

>> wb.uploadPlist(‘My AO’, pl)

downloadPlist

downloads the plist from the block. Input arguments are:

downloadPlist(blockName)

blockName - the name of the block to download the plist from

For example,

>> pl = wb.downloadPlist(‘My AO’);

appendPlist

appends a plist to the existing plist on the given block. Input arguments are

appendPlist(blockname, pl)

blockname - the name of the block
pl - the plist to append

For example,

>> wb.appendPlist(‘My AO’, pl);

appendParam

appends a parameter to the plist of the named block. The input arguments are

appendParam(blockname, param)
appendParam(blockname, key, value)

blockname - the name of the block to append this parameter to
param - a parameter object
key,value - key/value pair to append

For example,

>> wb.appendParam(‘My AO’, ‘name’, ‘bob’);

moveBlock

moves a block to a new location. The input arguments are:

moveBlock(blockName, x,y)

blockName - the name of the block to move
x - the new X coordinate
y - the new Y coordinate

Programatic execution control

Pipelines can be executed programatically using the following commands:

reset

Reset the pipeline to the start. Call

>> wb.reset

stepForward

Executes the current ready blocks and steps the pipeline forward one step. Call:

>> wb.stepForward

stepBackwards

Step backwards one step. Call:

>> wb.stepBackwards

skipForward

Moves the block cursor forwards one step without executing the current ready blocks. Call:

>> wb.skipForward

runPipeline

Reset the pipeline and run it from the beginning. Call:

>> wb.runPipeline