pyJMT Getting Started Guide

James Stadler

July 2023

1 Introduction

The pyJMT library is a Python wrapper for the Java Modelling Tools (JMT) suite, which is used for performance evaluation of systems using queueing network models. This guide will help you get started based on code examples. This guide is not exhaustive, and more detail of available objects and methods can be found here. Some knowledge of JMT is useful and great care has been made for this tool to be as intuitive as possible for users of JMT.

2 Library Import

First, install and then import the necessary libraries:

```
pip install pyJMT
```

```
import pyJMT as jmt
```

3 Creating Network Models

Network models, Nodes and job classes are represented by variables, which we then link and set parameters for to build our representation of a queueing network.

3.1 Example

Here's an example of a simple model:

```
import pyJMT as jmt

def gettingStarted():
    model = jmt.Network("gettingStarted")
    source = jmt.Source(model, "source")
    queue = jmt.Queue(model, "queue", jmt.SchedStrategy.FCFS)
    sink = jmt.Sink(model, "sink")

    oclass = jmt.OpenClass(model, "oclass")
    queue.setService(oclass, jmt.Exp(0.5))
    source.setArrival(oclass, jmt.Exp(1))

    model.addLinks([(source, queue), (queue, sink)])
    model.printResults()



if __name__ == '__main__':
    gettingStarted()
```

The gettingStarted function creates a model with one open class, one source, one queue, and one sink. The source creates jobs of the open class with an exponential interarrival time with a mean of 1. The queue services jobs of the open class also with an exponential service time with a mean of 0.5. The queue uses a first-come-first-served (FCFS) scheduling strategy. The results on the simulation are then printed to console.

4 Further Examples

In depth documentation can be found at here, including all available options for routing strategies, queuing strategies, metrics and drop strategies. This code is available on GitHub. A full list of supported functions can be found on the last page.

4.1 Job Classes

pyJMT supports open and closed job class types. Open classes require a network and name in their creation, and for a source to be set as their arrival node with a distribution. Closed classes require a network, a name, a population and a node to originate from. Both class types can optionally take in a priority value. Below is an example of class creation and use.

```
import pyJMT as jmt

def example_classes():
    model = jmt.Network("example classes")
    source = jmt.Source(model, "source")
    queue = jmt.Queue(model, "queue", jmt.SchedStrategy.FCFS)
    sink = jmt.Sink(model, "sink")

    # An open class with priority of 2
    openclass = jmt.OpenClass(model, "open class", 2)

    # A closed class with population of 10 originating from queue
    # with a default priority (0)
    closedclass = jmt.ClosedClass(model, "closed class", 10, queue)

    source.setArrival(openclass, jmt.Exp(1))

    # Open the model
    model.jsimgOpen()
```

4.2 Linking

Nodes in a model must be linked for jobclasses to transition between them. Below is an example of how to link together nodes as well as remove links.

```
import pyJMT as jmt

def example_linking():
    # Basic model
    model = jmt.Network("linking")
    source = jmt.Source(model, "source")
    queue = jmt.Queue(model, "queue", jmt.SchedStrategy.FCFS)
    sink = jmt.Sink(model, "sink")

    openclass = jmt.OpenClass(model, "open class")
    source.setArrival(openclass, jmt.Exp(1))
    queue.setService(openclass, jmt.Exp(1))

    # Add serial linking from source to sink through queue
    model.addLink(source, queue)
    model.addLink(queue, sink)

    # Remove linking added above
    model.removeLink(source, queue)
    model.removeLink(queue, sink)

    # Add links back in one line
    model.addLinks([(source, queue), (queue, sink)])
    # Remove links again in one line
    model.removeLinks([(source, queue), (queue, sink)])

    # Open the model
    model.jsimgOpen()
```

4.3 Service

The set of available service distributions can be seen here. These distributions are used to set the arrival time distributions of sources as well as the service time distributions of queues. Both queues and sources must have these values set. Below is an example of their use.

```
import pyJMT as jmt

def example_services():
    # Basic model with links
    model = jmt.Network("example services")
    source = jmt.Source(model, "source")
    queue = jmt.Queue(model, "queue", jmt.SchedStrategy.FCFS)
    sink = jmt.Sink(model, "sink")
```

```
    openclass1 = jmt.OpenClass(model, "open class 1")
    openclass2 = jmt.OpenClass(model, "open class 2")

    source.setArrival(openclass1, jmt.Cox(1.0, 2.0, 0.5))
    source.setArrival(openclass2, jmt.Exp(1))
    queue.setService(openclass1, jmt.Normal(2.0, 1.0))
    queue.setService(openclass2, jmt.ZeroServiceTime())
    queue.setNumberOfServers(3)

    model.addLinks([(source, queue), (queue, sink), (queue, queue)])

    # Open the model
    model.jsimgOpen()
```

4.4 Routing

All nodes except Sinks have options available for their routing. By default the routing is set to random, but this can be changed. A full list of available routing strategies can be found here. ClassSwitch and Probabilistic routing require their values to be set for each job class towards each linked node. Below is an example of a simple routing strategy being set as well as probabilistic and class switch routing setup.

```
import pyJMT as jmt

def example_routing():
    # Basic model with links
    model = jmt.Network("routing")
    source = jmt.Source(model, "source")
    queue = jmt.Queue(model, "queue", jmt.SchedStrategy.FCFS)
    sink = jmt.Sink(model, "sink")

    openclass1 = jmt.OpenClass(model, "open class 1")
    openclass2 = jmt.OpenClass(model, "open class 2")

    source.setArrival(openclass1, jmt.Exp(1))
    source.setArrival(openclass2, jmt.Exp(1))
    queue.setService(openclass1, jmt.Exp(1))
    queue.setService(openclass2, jmt.ZeroServiceTime())

    model.addLinks([(source, queue), (queue, sink), (queue, queue)])

    # Class 1 jobs have 100 percent chance of going to queue
    source.setRouting(openclass1, jmt.RoutingStrategy.PROB)
```

```
    source.setProbRouting(openclass1, queue, 1.0)

    # Jobs of class 1 change to class 2 half the time and return to start of queue,
    # The other half of the time they continue to the sink as class 1 jobs
    queue.setRouting(openclass1, jmt.RoutingStrategy.CLASSSWITCH)
    queue.setClassSwitchRouting(openclass1, openclass2, queue, 0.5, 1.0)
    queue.setClassSwitchRouting(openclass1, openclass1, sink, 0.5, 1.0)

    # Jobs of class 2 go to queue or sink in a round robin strategy
    queue.setRouting(openclass2, jmt.RoutingStrategy.RROBIN)

    # Open the model
    model.jsimgOpen()
```

4.5 Queues

Queues can have their capacity, drop strategy (if applicable) and queuing strategy modified. Below is an example of how to change each of these parameters.

```
import pyJMT as jmt

def example_queues():
    # Basic model with links
    model = jmt.Network("example services")
    source = jmt.Source(model, "source")
    queue = jmt.Queue(model, "queue", jmt.SchedStrategy.FCFS)
    sink = jmt.Sink(model, "sink")

    openclass1 = jmt.OpenClass(model, "open class 1")
    openclass2 = jmt.OpenClass(model, "open class 2")

    source.setArrival(openclass1, jmt.Cox(1.0, 2.0, 0.5))
    source.setArrival(openclass2, jmt.Exp(1))
    queue.setService(openclass1, jmt.Normal(2.0, 1.0))
    queue.setService(openclass2, jmt.ZeroServiceTime())
    queue.setNumberOfServers(3)

    # Set additional queue parameters
    queue.setStrategy(jmt.SchedStrategy.FCFS_PRIORITY)
    queue.setCapacity(10)
    queue.setDropRule(jmt.DropStrategy.WAITING_QUEUE)

    model.addLinks([(source, queue), (queue, sink)])

    # Open the model
```

```
    model.jsimgOpen()
```

4.6 Forks and Joins

Forks and joins can be added in a straightforward way, with a single additional method to set the number of tasks to be generated towards each link for forks. Below is an example of a simple network with a fork and a join.

```
import pyJMT as jmt

def example_forksjoins():
    # Basic model with links
    model = jmt.Network("example forks and joins")
    source = jmt.Source(model, "source")
    queue1 = jmt.Queue(model, "queue 1", jmt.SchedStrategy.FCFS)
    queue2 = jmt.Queue(model, "queue 2", jmt.SchedStrategy.FCFS)
    fork = jmt.Fork(model, "fork")
    join = jmt.Join(model, "join")
    sink = jmt.Sink(model, "sink")

    openclass1 = jmt.OpenClass(model, "open class 1")

    source.setArrival(openclass1, jmt.Cox(1.0, 2.0, 0.5))
    queue1.setService(openclass1, jmt.Normal(2.0, 1.0))
    queue1.setNumberOfServers(3)
    queue2.setService(openclass1, jmt.Exp(1.0))
    queue2.setNumberOfServers(2)

    fork.setTasksPerLink(1)
    model.addLinks([(source, fork), (fork, queue1), (fork, queue2),
                    (queue1, join), (queue2, join), (join, sink)])

    # Open the model
    model.jsimgOpen()
```

4.7 Adding metrics

By default a standard set of metrics are added to all nodes in the model, which is Number of Customers, Utilization, Response Time, Throughput and Arrival Rate. These default metrics can be switched on or off, and additonal metrics can be added or removed. Below is an example of a simple network with disabled default metrics and 2 additional metrics added. Additioanl metrics can also be removed similarly.

```
import pyJMT as jmt

def example_metrics():
    # Basic model with links
    model = jmt.Network("example metrics")
    source = jmt.Source(model, "source")
    queue = jmt.Queue(model, "queue", jmt.SchedStrategy.FCFS)
    sink = jmt.Sink(model, "sink")

    openclass1 = jmt.OpenClass(model, "open class 1")
    openclass2 = jmt.OpenClass(model, "open class 2")

    source.setArrival(openclass1, jmt.Cox(1.0, 2.0, 0.5))
    source.setArrival(openclass2, jmt.Exp(1))
    queue.setService(openclass1, jmt.Normal(2.0, 1.0))
    queue.setService(openclass2, jmt.ZeroServiceTime())
    queue.setNumberOfServers(3)

    model.addLinks([(source, queue), (queue, sink), (queue, queue)])

    # Switch off default metrics
    model.useDefaultMetrics(False)
    # Add total customers metric to whole system for class 1
    model.addMetric(openclass1, model, jmt.Metrics.NUM_CUSTOMERS)
    # Add response time metric to queue for class 2
    model.addMetric(openclass2, queue, jmt.Metrics.RESPONSE_TIME)

    # Open the model
    model.jsimgOpen()
```

4.8 Running simulations and getting results

pyJMT can generate 2 different types of files, firstly .jsimg files which are directly openable in JMT, secondly results files (.jsim) which is the results output by running a simulation on a .jsimg. From a results file pyJMT can either print the results or return a dictionary of the results.

```
import pyJMT as jmt

def example_saveandload():
    # Basic model with links
    model = jmt.Network("example save and load")
    source = jmt.Source(model, "source")
```

```
queue = jmt.Queue(model, "queue", jmt.SchedStrategy.FCFS)
sink = jmt.Sink(model, "sink")

openclass1 = jmt.OpenClass(model, "open class 1")
openclass2 = jmt.OpenClass(model, "open class 2")

source.setArrival(openclass1, jmt.Cox(1.0, 2.0, 0.5))
source.setArrival(openclass2, jmt.Exp(1))
queue.setService(openclass1, jmt.Normal(2.0, 1.0))
queue.setService(openclass2, jmt.ZeroServiceTime())
queue.setNumberOfServers(3)

model.addLinks([(source, queue), (queue, sink), (queue, queue)])

# NOTE: ALL SAVED FILES GO TO ./output_files
# save the current model as a .jsimg (can be opened in JMT)
model.saveNamedJsimg("model")
# generate a results file (.jsim) from the current model called "model"
# (does not require a saved JSIMG first)
model.saveResultsFileNamed("modelResults")
# print the results from the current model
# (does not require a saved results file beforehand)
model.printResults()
# return a dictionary of the results from the current model
# (does not require saving)
results_dict = model.getResults()

# generate a results file from a .jsimg called "model"
jmt.saveResultsFromJsimgFile("modelResults")
# print the results from a results file (.jsim) called "model"
# (does not require a saved results file beforehand)
jmt.printResultsFromResultsFile("modelResults")
# return a dictionary of the results from a results file called "model"
results_dict = jmt.getResultsFromResultsFile("modelResults")

# Open the model
model.jsimgOpen()
```

Table 1: Supported JMT features

JMT Feature	**Support**	**Notes**
Distributions	Full	Deterministic, Exponential, Erlang, Gamma, Hyperexponential, Coxian, Logistic, Normal, Pareto, Uniform, Replayer, Weibull, Disabled
Classes	Full	Open class, Closed class, Class priorities
Metrics	Full	Number of customers, Residence Time, Throughput, Response Time, Throughput per sink, Utilization, Arrival Rate, Drop rate, Response time per sink, Power
Nodes	Full	Source, Sink, Delay, Queue, Router, ClassSwitch, Logger, Finite Capacity Region
Routing	Full	Random, Round Robin, Join the Shortest Queue, Shortest Response Time, Least Utilization, Fastest Service time, Probabilities, Class Switch Routing
Scheduling	Full	FCFS, FCFS-PR, LCFS, LCFS-PR, SIRO (Random), SJF, SEPT, LJF, LEPT, PS, DPS, GPS, LPS, SRPT, HOL
Distributions	No	Phase-Type, Burst (MAP), Burst (MMPP2), Burst (General), Disabled
Nodes	No	Scaler, Zero Service Time, Semaphore, Place, Transition
Routing	No	Load Dependent
Scheduling	No	Polling
Mechanisms	No	Load Dependence, Retrial, Impatience, Setup times