*Proceedings of the 2013 Winter Simulation Conference*
*R. Pasupathy, S. -H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.*

# SUPPORTING A MODELING CONTINUUM IN SCALATION: FROM PREDICTIVE ANALYTICS TO SIMULATION MODELING SUPPLEMENT

John A. Miller[1], Michael E. Cotterell[1], Stephen J. Buckley[2]

[1]Department of Computer Science
University of Georgia
Athens, GA 30602, USA

[2]IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598, USA

**This supplement to the paper provides all figures and substantial code listings which would not fit into the paper due to page limitations. There is also an appendix which provides an overview of the ScalaTion, an integrated environment supporting predictive analytics, simulation modeling and optimization.**

**Permalink:** **http://www.cs.uga.edu/~jam/scalation/apps/simopt**

**Roadmap:** The supplement is organized as follows: Section 1 provides some code listings for Section 3 (SIMULATION MODELING) of the main paper, Section 2 provides tables, figures, and code listings for Section 5 (APPLICATION TO HEALTHCARE) of the main paper, and Section 3 provides figures for Section 6 (APPLICATION TO SUPPLY CHAIN MANAGEMENT). After the references, an appendix is included witch provides an overview of ScalaTion.

**Note:** In many cases, text from the main paper is included and or extended in order to provide context for the figures, tables, and code listings contained in this supplement.

## 1 SIMULATION MODELING EXAMPLES

### 1.1 Dynamics Model

System dynamics models were recently added to DeMO, since hybrid models that combine continuous and discrete aspects are becoming more popular. In this section, modeling the flight of a golf ball is reconsidered. Let the response vector $\mathbf{y} = [y_0\ y_1]$ where $y_0$ indicates the horizontal distance traveled, while $y_1$ indicates the vertical height of the ball. Future positions $\mathbf{y}$ depends on the current position and time $t$. Using Newton's Second Law of Motion, $\mathbf{y}$ can be estimated by solving a system of Ordinary Differential Equations (ODEs) such as $\dot{\mathbf{y}} = f(\mathbf{y}, t),\ \mathbf{y}(0) = \mathbf{y_0}$. The `Newtons2nd` object uses the Dormand-Prince ODE solver to solve this problem. More accurate models for estimating how far a golf ball will carry when struck by a driver can be developed based on inputs/factors such as club head speed, spin rate, smash factor, launch angle, dimple patterns, ball compression characteristics, etc. There have been numerous studies of this problem, including (Barber III 2011). Here are some slightly modified excerpts from the `Newtons2nd` object provided as an application example in ScalaTion:

```
// Golf Ball Flight Dynamics Model
val n  = 100                          // maximum number of time points
val tm =   5.                         // simulate for a maximum of tm sec
val g  =   9.80665                    // gravitational force in m/sec^2
val m  =  45.93                       // mass of a golf ball in grams
val aa =  15.00                       // launch angle in degrees
val ss = 100.00                       // swing speed in miles/hour
val sf =   1.49                       // smash factor
val s  = ss * sf * 1609.344 / 3600    // initial ball speed in m/sec
val a  = aa * Pi / 180.               // launch angle in radians
val y0 = VectorD (0., 0.)             // initial position (y_0, y_1)
val v0 = VectorD (s*cos(a), s*sin(a)) // initial velocity at t0

// define the system of Ordinary Differential Equations (ODEs)
def dy1_dt (t: Double, y: VectorD) = v0(0)              // ODE 1
def dy2_dt (t: Double, y: VectorD) = v0(1) - g * t      // ODE 2
val odes: Array [Derivative] = Array (dy1_dt, dy2_dt)
```

As you can see, defining the system of ODEs in ScalaTion is as easy as defining Scala functions using the appropriate types and placing them in an `Array`. Alternatively, the following syntax utilizing lambda expressions could also have been used:

```
// define the system of Ordinary Differential Equations (ODEs)
val odes = Array [Derivative] (
    (t: Double, y: VectorD) => v0(0),                  // ODE 1
    (t: Double, y: VectorD) => v0(1) - g * t           // ODE 2
) // odes
```

As mentioned earlier, the Dormand-Prince ODE solver is used to solve this problem. ScalaTion provides an implementation of this in the `DormandPrince` object. Below, the constant time step `dt` is defined and variables for the next time point to examine `t` and the current ball position `y` are initialized.

```
val dt = tm / n                               // time step
var t  = dt                                   // next time point to examine
var y  = VectorD.ofDim (2)                    // current ball position
```

Using a breakable loop, the flight of a golf ball is simulated below by iteratively integrating the ODEs from impact until either the ball hits the ground or the maximum number of iterations `n` is reached. During each iteration, the new position is calculated via `DormandPrince.integrateV` and the time step `t` is advanced.

```
breakable { for (i <- 1 to n) {
   y = DormandPrince.integrateV (odes, y0, t)  // compute new position
   if (y(1) < 0.) break                        // quit after hitting ground
   println ("> at t = %4.1f, y = %s".format (t, y))
   t += dt
}} // for
```

## 2 APPLICATION TO HEALTHCARE EXAMPLES

**Code snippets in this section can be found at the following link: https://code.google.com/p/scalation/source/browse/branches/mec-wsc13/examples/simopt/**

In the healthcare domain, one problem to be addressed for emergency departments/urgent care facilities is that of staffing. The solutions provided below are simplified to better illustrate the techniques. For more information on problems of this type, please see (Tan et al. 2012). Given an estimated demand, how many of various types of staff members should be hired, i.e., how many triage nurses, registered nurses, nurse practitioners, doctors and administrative clerks should be hired. The model includes $l = 2$ types of patients (regular and severe) and $m = 5$ types of employees.

Table 1: Model Definitions

| Variable | Obtained | Description | Units |
|---|---|---|---|
| $\lambda_k$ | input | arrival rate for patients of type $k$ | $\text{hr}^{-1}$ |
| $\mu_{jk}$ | input | service rate at resource $j$ for patients of type $k$ | $\text{hr}^{-1}$ |
| $f_k$ | input | fee charged to patients of type $k$ | \$ |
| $d$ | input | patients dis-utility of waiting | \$ / hr |
| $s_j$ | input | salary/wage for staff of type $j$ | \$ / hr |
| $x_j$ | optimize | staffing level for type $j$ employees | none |
| $n$ | output | treatment rate for patients | $\text{hr}^{-1}$ |
| $w$ | output | waiting time for patients | hr |
| $c$ | $\mathbf{s} \cdot \mathbf{x}$ | operating cost | \$ / hr |
| $r$ | $\mathbf{f} \cdot \mathbf{n}$ | revenue for patient service | \$ / hr |
| $p$ | $r - c$ | net profit | \$ / hr |
| $u$ | $p - dnw$ | overall utility | \$ / hr |

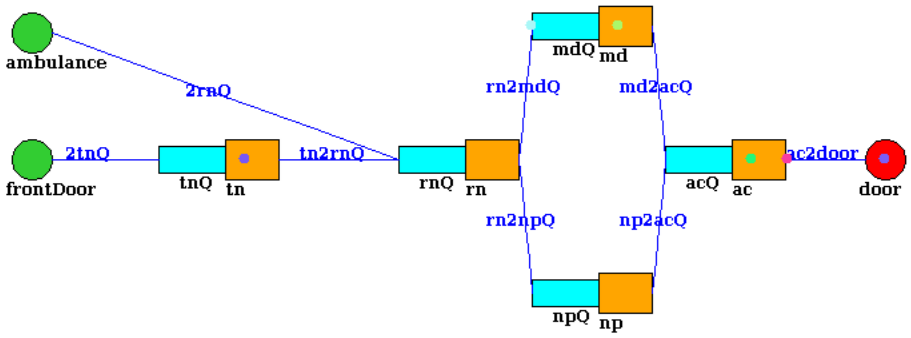


Figure 1: Screenshot of the Animation Produced by ScalaTion.

The goal is to maximize a utility function based on profit as well as patient satisfaction that factors in a dis-utility proportional to patient waiting times. The optimization problem may be formulated as follows:

$$\textbf{max } u(\mathbf{x}) \textbf{ subject to } \mathbf{x} \in \mathbb{Z}_+^m$$

This is an Integer Nonlinear Programming Problem (INLP) where, unless assumptions/approximations are made, there is no closed-form expression for the objective function $u(\mathbf{x})$. For this modeling/optimization

problem, the following three techniques are utilized: Process-Interaction Simulation Models, Queueing Networks and Multiple Linear Regression.

### 2.1 General Model Parameters

Model parameters used for the various models are given below (see also https://code.google.com/p/scalation/source/browse/branches/mec-wsc13/examples/simopt/ERParams.scala):

```
val lambda = VectorD (.1, .05, .0, .0, .0)               // arrival rates
val mu     = MatrixD ((2, 5)), .25, .16, .05, .10, .16,  // service rates
                               .20, .15, .04, .12, .15))

val d = 1.0                           // patients dis-utility of waiting
val s = VectorD (10, 50, 120, 80, 8) // salary/wage for each staff member
val f = VectorD (80, 150)             // fee charged to patients (low, high)
```

The values in `lambda` and `mu` are rates and not means. For the some of the models, these rates are used directly (e.g., in the steady-state calculations in the queueing model), and in other models, exponential distributions are created using the corresponding mean value. A `VectorD` containing rates can easily be converted to a `VectorD` containing the corresponding mean values invoking the vector's `reciprocal` function.

### 2.2 Process-Interaction Simulation Model

#### 2.2.1 Simulation Model

The source code for the `ERModel` class is too large to be included in this supplement. However, it can be found at https://code.google.com/p/scalation/source/browse/branches/mec-wsc13/examples/simopt/ERModel.scala.

#### 2.2.2 Optimization Example

Using the `ERModel` class, it is possible to formulate the objective function for the INLP problem described for the Healthcare application. In the listing below, the function `u_p` will return the overall utility calculation after gathering statistics from a an instance of the `ERModel` class given a staffing level provided in the `x` vector. After an instance of a valid optimizer created using this function, an initial staffing level is chosen, the utility function is maximized, and the results are displayed. In this case, the `IntegerTabuSearch` class from the `scalation.maxima` package was chosen because it has the nice property that once a point is found to be sub-optimal, it is never visited again.

```
// Utility Function for Healthcare Application
// Uses the Process-Interaction Model
def u_p (x: VectorI): Double = {
    val m = new ERModel(x)              // create model
    val results = m.simulate()          // simulate and gather stats
    val w = m.sumMeanWaitingTimes()     // sum of average waiting times
    val n = VectorD (m.low, m.high)     // patients served of each type
    val c = x dot s                     // operating payroll cost
    val r = f dot n                     // revenue for patient service
    val p = r - c                       // net profit
    p - d * w * n.mean                  // return overall utility
} // u_p
```

```
val x0          = VectorI (1, 1, 1, 1, 1)             // starting point
val its         = new IntegerTabuSearch (u_p)         // setup optimizer
val (x, result) = its.maximize (x0)                   // get results
println ("x = %s; u_p value = %f".format(x, result))  // print results
```

The values in `c` and `r` are each the dot product (outer product) of two real-valued vectors. ScalaTion provides the `dot` operator to make this calculation concise and painless.

### 2.3 Queueing Network Model

#### 2.3.1 Jackson Queueing Network

The `ERQueueingModel` class is provided in the listing below. This queuing model relies on an instance of the `JacksonNet` class provided by ScalaTion. The following class computes the steady-state results for a M/M/c queueing network given a probability transition matrix `p`, the rates from `lambda` and `mu` as well as the staffing levels provided in the `x` vector.

```
// Queueing Model for Healthcare Application
class ERQueueingModel (x: VectorI)
{
    //                        DESTINATIONS                          S
    //                        TN    RN    MD    NP    AC          O
    val p = MatrixD ((5, 5), 0.00, 1.00, 0.00, 0.00, 0.00, // TN U
                             0.00, 0.00, 0.25, 0.75, 0.00, // RN R
                             0.00, 0.00, 0.00, 0.00, 1.00, // MD C
                             0.00, 0.00, 0.00, 0.00, 1.00, // NP E
                             0.00, 0.00, 0.00, 0.00, 0.00) // AC S

    val jqn = new JacksonNet (p, lambda, mu(0), x) // create model

    // calculate expected number and waiting time in the system
    val n   = (0 until jqn.m).map(i => jqn.nQueue(i) + jqn.rho(i) * x(i)).sum
    val w   = (0 until jqn.m).map(i => jqn.nQueue(i) / jqn.lambda(i)).sum

} // ERQueueingModel
```

Note that since this a model of a M/M/c queueing network, service times are exponentially distributed according to the first row vector in `mu`. The values `n` and `w` are both calculated by mapping each node's index value to an appropriate calculation involving that index value, then taking the sum over the resulting mapped vector. Scala's `map` function was used to avoid the clutter of using multiple for loops.

#### 2.3.2 Optimization Example

Using the `ERQueueingModel` class, it is possible to formulate the objective function for the INLP problem described for the Healthcare application. In the listing below, the function `u_q` will return the overall utility calculation after gathering statistics from a an instance of the `ERQueueingModel` class given a staffing level provided in the `x` vector. Just like the previous optimization example, the `IntegerTabuSearch` class was chosen as the optimizer.

```
// Utility Function for Healthcare Application
// Uses the Queueing Model
```

```
def u_q (x: VectorI): Double = {
    val m = new ERQueueingModel (x) // create queueing model
    val c = x dot s                 // operating payroll cost
    val r = f(0) * m.n              // revenue for patient service
    val p = r - c                   // net profit
    p - d * m.w * m.n               // return overall utility
} // u_q

val x0          = VectorI (1, 1, 1, 1, 1)             // starting point
val its         = new IntegerTabuSearch (u_q)         // setup optimizer
val (x, result) = its.maximize (x0)                   // get results
println ("x = %s; u_q value = %f".format(x, result))  // print results
```

Since this formulation of the model counts the expected number of patients in the system and the number total number of each type of patient, only the first value in `f` was used to calculate the revenue in `r`.

### 2.4 Multiple Linear Regression Model

#### 2.4.1 Regression Model

The source code for the `ERRegressionModel` class is too large to be included in this supplement. However, it can be found at https://code.google.com/p/scalation/source/browse/branches/mec-wsc13/examples/simopt/ERRegressionModel.scala.

#### 2.4.2 Optimization Example

First an existing function needs to be chosen in order provide data for the design points used for regression. This can be done by creating a function that utilizes one of our previously defined utility functions. Here is an example of such a function, using the utility function that was defined to use the Process-Interaction Simulation model:

```
def u_r (x: VectorD): Double = u_p (x.toVectorI)
```

The function above could easily be modified to incorporate historical data. Suppose there exists a function `h` that takes the input vector `x` and returns a `Statistic` object corresponding to the statistics for empirical observations of the real-world system when the staffing levels match the staffing level provided by `x`. Now, the utility function for regression can be defined as follows:

```
def u_r (x: VectorD): Double = {
    val stat = h (x.toVectorI)                          // get empirical stats
    if (stat != null && stat.interval <= 0.1) stat.mean // try to use stats
    else u_p (x.toVectorI)                              // otherwise simulate
} // u_r
```

Now, applying an optimization routine in order to maximize utility using the regression model is similar to the previous examples.

```
val m           = new ERRegressionModel (u_r)           // regression model
val x0          = VectorI (1, 1, 1, 1, 1)               // starting point
val its         = new IntegerTabuSearch (m.eval)        // setup optimizer
val (x, result) = its.maximize (x0)                     // get results
println ("x = %s; m.eval value = %f".format(x, result)) // print results
```

The `ERRegressionModel` class partitions the overall surface into regions. Multiple Linear Regression is used to fit the response surface to design points that are distributed within each region. When constructing the design matrix for a region, the response for a particular design point is determined by `u_r`. If historical data is being considered, then statistics derived from empirical observations are used so long as they meet certain confidence criteria (e.g., the confidence interval half-width is less than some threshold). When historical statistics are either poor or simply do not exist for a particular staffing level, then a simulation is run in order to provide those statistics.

## 3 SUPPLY CHAIN MANAGEMENT FIGURES

A wide variety of time-dependent predictive analytics techniques are used in supply chain management to forecast product demand (Box and Jenkins 1976). As shown in Figure 2, forecasts of product demand feed the overall supply chain process, whose goal is to provide inventory to satisfy demand on a continuing basis. Simulation is often used to assess whether a supply chain will truly satisfy demand in the presence of a variety of uncertainties such as forecast error, supplier lead time, manufacturing lead time, and manufacturing yield.

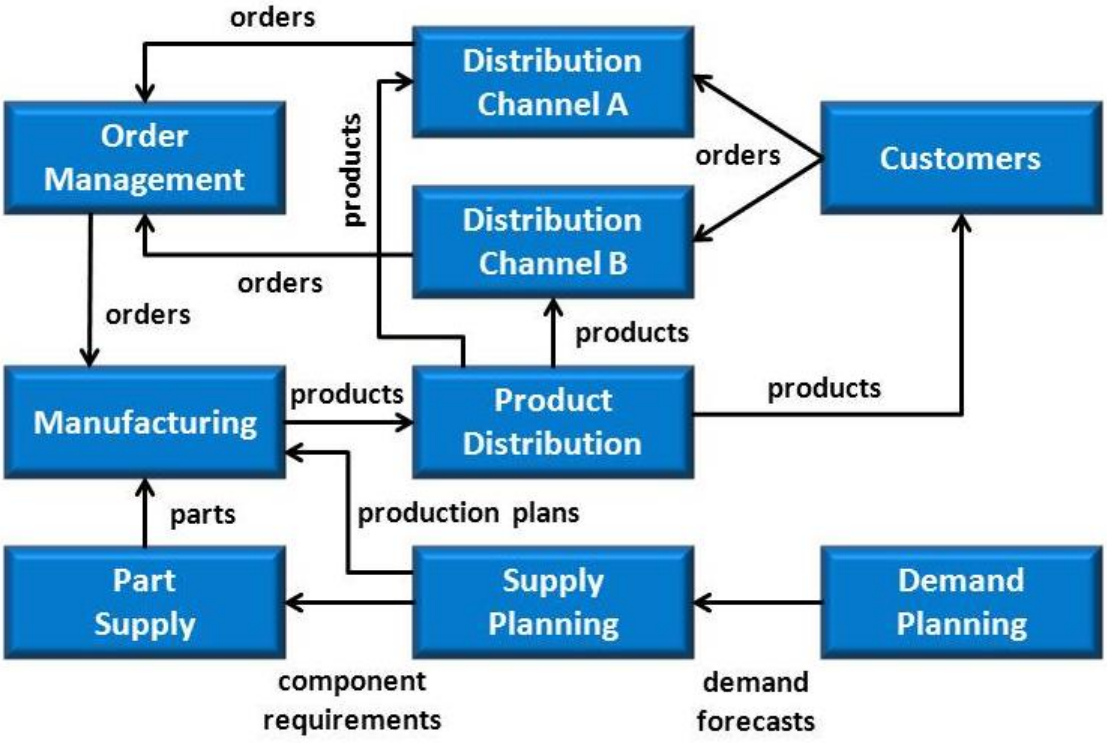


Figure 2: A Model of Supply Chain Processes

**IBM Pandemic Business Impact Modeler**: In 2006, IBM developed a simulation model to understand how a pandemic might impact a manufacturing companys employees and business performance (Chen-Ritzo et al. 2007). As shown in Figure 3, the model comprised six integrated sub-models to examine epidemiological, behavioral, economic, infrastructure, value chain, and financial aspects of the IBM ecosystem. The sub-models were constructed using a combination of system dynamics simulation, time step simulation, and linear programming.

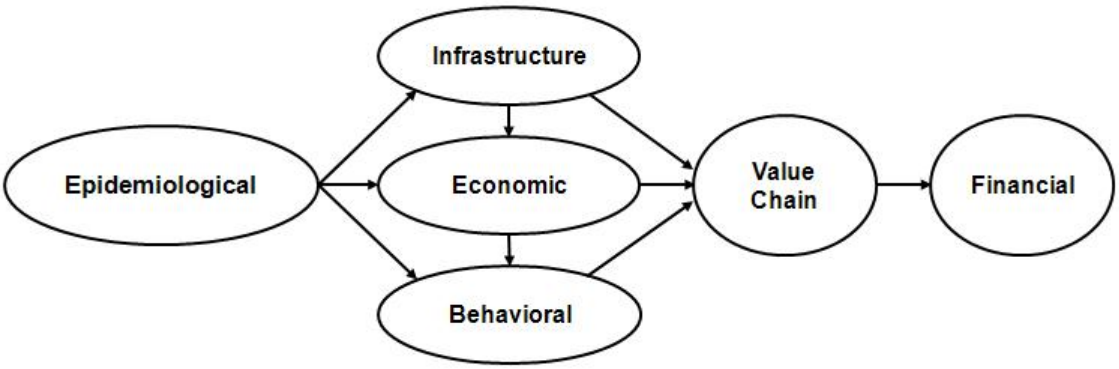


Figure 3: Simulation Flow in the IBM Pandemic Business Impact Modeler

**IBM Europe PC Study**: $40M of savings were realized per year by removing many distribution centers and transshipment points, and changing the manufacturing execution strategy from Build To Forecast to a variant of Build To Order (see Figure 4).

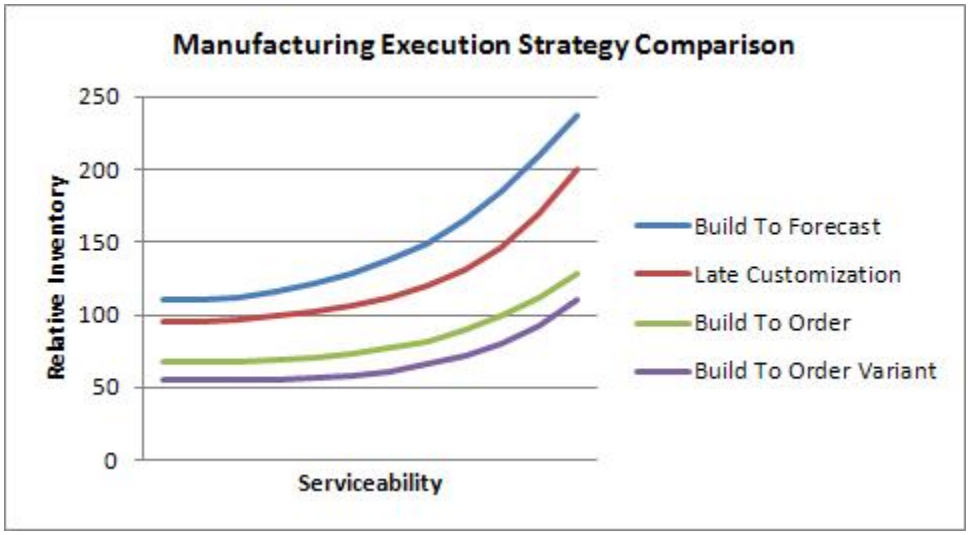


Figure 4: Analysis of Manufacturing Execution Strategies in the IBM Europe PC Study

## REFERENCES

Barber III, J. 2011". "Golf Ball Flight Dynamics". Technical report.

Box, G., and G. Jenkins. 1976. *Time Series Analysis: Forecasting and Control*. Upper Saddle River, NJ, USA: Holden-Day.

Chen-Ritzo, C., L. An, S. Buckley, P. Chowdhary, T. Ervolina, N. Lamba, Y. Lee, and D. Subramanian. 2007. "Pandemic Business Impact Modeler". In *Proceedings of the 2007 INFORMS Simulation Society Research Workshop*: The Society for Computer Simulation.

Tan, W.-C., P. J. Haas, R. L. Mak, C. A. Kieliszewski, P. G. Selinger, P. P. Maglio, S. Glissman, M. Cefkin, and Y. Li. 2012. "Splash: A Platform for Analysis and Simulation of Health". In *Proceedings of the 2nd ACM SIGHIT Symposium on International Health Informatics*, 543–552. ACM.

## APPENDIX - ScalaTion Overview

ScalaTion, a Scala-based Domain-Specific Language (DSL) and framework, serves a testbed for exploring a modeling continuum that includes analytics, simulation and optimization.

### Availability

ScalaTion can be downloaded using the URLs available at http://www.cs.uga.edu/~jam/scalation/README.html.

### Documentation and Related Papers

- John A. Miller, Jun Han and Maria Hybinette, "Using Domain Specific Languages for Modeling and Simulation: ScalaTion as a Case Study," In *Proceedings of the 2010 ACM/IEEE Winter Simulation Conference (WSC'10)*, Baltimore, Maryland (December 2010) pp. 741–752.
- Michael E. Cotterell, John A. Miller, Tom Horton, "Unicode in Domain-Specific Programming Languages for Modeling & Simulation: ScalaTion as a Case Study," In *Arxiv preprint arXiv:1112.175* (December 2011) pp. 1–10.
- Michael E. Cotterell, John A. Miller, Jun Han and Tom Horton, "Extending ScalaTion, a Domain-Specific Language for Modeling & Simulation, for Simulation Optimization" In *Proceedings of the AlaSim International Modeling and Simulation Conference & Exhibition (AlaSim'12)*, Huntsville, Alabama (May 2012) pp. 1–1.
- Yung Long Li, "Evaluation of Parallel Implementations of Dense and Sparse Matrices for the ScalaTion Library," Technical Report, University of Georgia (December 2012) pp. 1–60.
- ScalaDoc API for ScalaTion: http://www.cs.uga.edu/~jam/scalation/doc/

### Source Packages

| Package Name | Description |
|---:|---|
| `scalation.util` | A package of utilities needed by the other packages. |
| `scalation.math` | A package of mathematical objects and operations needed for analytics, simulation and optimization. |
| `scalation.linalgebra` | A package of implementations for linear algebra (e.g., for vectors and matrices). Vectors and matrices of real (`Double`) and `Complex` numbers are supported. |
| `scalation.linalgebra_gen` | A package of generic implementations for linear algebra (e.g., for vectors and matrices). Vectors and matrices of types implementing `Numeric` can be instantiated. |
| `scalation.calculus` | A package for computing derivatives, gradients and Jacobians. |
| `scalation.random` | A package for random numbers and random variates (Normal distribution, etc.) |
| `scalation.stat` | A package of statistical objects and operations needed for simulation, including implementations for summary statitistics and ANOVA. |
| `scalation.scala2d` | A scala version of Java 2D. |
| `scalation.plot` | A package for displaying basic plots and histograms. |
| `scalation.animation` | A general purpose 2D animation engine. |

*Continued on the next page...*

| Package Name | Description |
|---|---|
| scalation.minima | A package supporting simulation optimization (minimization). |
| scalation.maxima | A package supporting simulation optimization (maximization). |
| scalation.analytics | A package supporting analytics, including regression, time series analysis and clustering. |
| scalation.graphalytics | A package supporting graph analytics, including shortest path, etc. |
| scalation.metamodel | A package supporting simulation metamodeling, espcially for optimization. |
| scalation.queueingnet | A package supporting queueing network models. |
| scalation.dynamics | A simulation engine for systems dynamics (continuous simulation), which includes general-purpose Ordinary Differential Equation (ODE) solvers. |
| scalation.dynamics_pde | A simulation engine for systems dynamics (continuous simulation), which includes category-specific Partial Differential Equation (PDE) solvers. |
| scalation.activity | A simulation engine for activity oriented models such as Petri Nets. |
| scalation.event | A simulation engine for event oriented models such as Event Graphs. |
| scalation.process | A simulation engine for process oriented models such as Process-Interaction Models. |
| scalation.state | A simulation engine for state oriented models such as Markov Chains. |

**Apps Packages**

| Package Name | Description |
|---|---|
| activity | Example models for activity oriented models such as Petri Nets. |
| analytics | Example analytics problems. |
| dynamics | Example models for systems dynamics (continuous simulation). |
| event | Example models for event oriented models such as Event Graphs. |
| game | Example simulation-oriented games. |
| montecarlo | Example Monte Carlo simulation. |
| optimization | Example optimization problems. |
| process | Example models for process oriented models such as Process-Interaction Models. |
| simopt | Example simulation optimization problems. |
| state | Example models for state oriented models such as Markov Chains. |