Georgia Tech : Course ISYE 6644

Learn & Implement SIMLIB

Using the book – "Simulation Language from the Simulation modeling and analysis" by Averill Law

Project By:

Jaykumar Kakkad (jayk@gatech.edu)

Rishikesh Gaikwad (rishikesh.gaikwad@gatech.edu)

Shivani Bhor (sbhor3@gatech.edu)

Contents

Summary

In this project, our goal is to understand how the SIMLIB language (in Chapter 2 of book by Law) works, implement SIMLIB in Python and provide a user guide and tutorial. SIMLIB provides useful functions and a event-driven framework that can be used to program discrete models. We implemented SIMLIB in python using the object oriented approach and dynamic storage using Linkedlist. Finally, we have provided a simple user guide and a tutorial for users.

SIMLIB provides an event-driven framework to program discrete models

SIMLIB is event-driven discrete simulation software. After studying the chapter 2, we found that all the basic operations related timing, event and random-variate generation are provided by SIMLIB in form of functions. This framework makes coding simulation models relatively easier. Once initialized, SIMLIB's main program just requires a logical sequence of timing routines and appropriate event routines.

Implementing SIMLIBPY using Object Oriented approach & Linkedlist

After studying the SIMLIB from the book, we proceeded to implement it in Python. Given the structure of storage of data in SIMLIB, we decided to use the object oriented approach in Python. We made three key computational decisions in Python - 1) Use of Linkedlist class as data structure, 2) Storing SIMLIB lists as array (list in python) of Linkedlist and 3) defining SIMLIB as a class so that all variables remain global. Please note that the User does not have to know these aspects.

We provide a simple user guide and tutorial to use the functionalities

In this report we provide simple user guide i.e. we summarize all the functions into a table and provide key instructions for users to write the main program. We also provide a tutorial by implementing Single server queue example using the SIMLIB functions.

Conclusion & learnings

Implementing SIMLIB helped us understand minor details of how each functions operate. While learning and implementing it, we realized that there are advantages and a few drawbacks of SIMLIB. It is open source and hence cost effective. Using object oriented approach, SIMLIB can be tailored to specific application and can be made more efficient than generic process interaction software. However, it has all the drawbacks of event-driven approach mainly higher programming cost, difficult to learn and detect errors. Our next step can be make use parallel & distributed programming concepts to make efficient even for complex applications.

Description of problem and background

SIMLIB was developed by Dr Petr Peringer at the BRNO University of Technology in 1991. Later multiple versions of SIMLIB were developed using new computing approaches.

Problem statement:

Related to understanding SIMLIB: Our objective was to understand the basic SIMLIB language as provided in the book by Prof Law. Firstly, we tried to answer few basic questions about the SIMLIB language

- What is the modeling approach?
- How does a generic event driven language work?
- What are key components and functions of SIMLIB?

Related to Implementing SIMLIB: Once we answered the above questions about SIMLIB, we decided to code it in Python. However, we ran into a few implementation questions/problems

- How to implement dynamic storage in Python?
- SIMLIB uses many global variables through use of pointers. How to we implement it in Python?
- How can we make a simple user guide and tutorial so as to make it easier for users to use SIMLIB?

In the next few sections of this report, we discuss below detail:

- How SIMLIB works?
- Computational decisions we made for implementing SIMLIB in Python
- Short user guide and Tutorial.
- Our conclusions and learning

Main Findings & Approach

In the below section we highlight our findings of how SIMLIB works. We also provide our computational approach to implementing SIMLIB in python. We then provide user guide and a tutorial example.

I) How SIMLIB Works?

SIMLIB is a software framework that helps code a discrete event Simulation models. The Main program uses the following components of the model of a system.

Key Components

- System state: Set of variables that describe a system at a point in time
- Simulation clock: Variable that provides current time
- Subroutines:
  - Event subroutine: Updates the system states when an event occurs
  - Timing subroutine: determines the next event and advances the simulation clock
  - Random generator subroutine: provides pseudo random numbers for specific distribution

Functions: We provide details of all the functions in the user guide section.

Main program:

The flowchart highlighted below describes the complete functioning of the main program in SIMLIB.

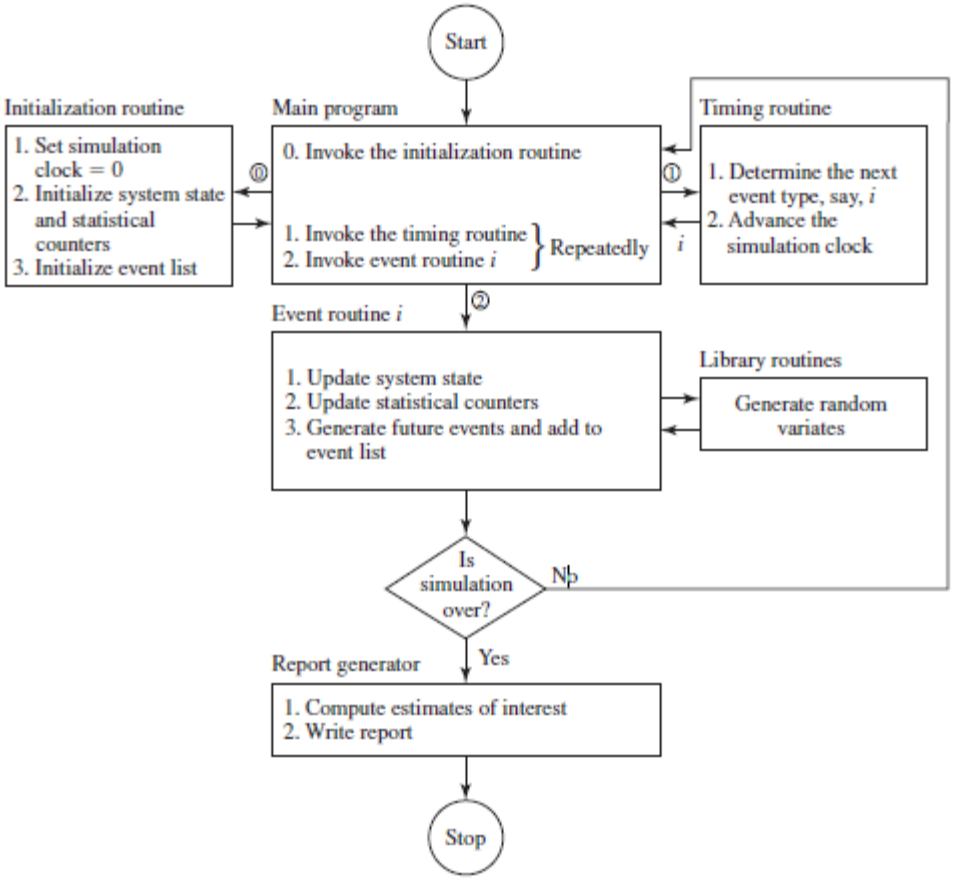
- Step 1: the user initializes the SIMLIB functions, variables and constants.
- Step 2: Invoke timing routine - it decides the event and advances the clock
- Step 3: Execute event routine based on the output of timing routine

IF (the program has **NOT** reached the end condition):

Repeat Step 2 and 3,

Else, generate report and end the program

Figure: Flowchart of how SIMLIB works



Source: *Simulation modeling and analysis by Averill Law*

II) Computational ideas & Object oriented approach in Python:

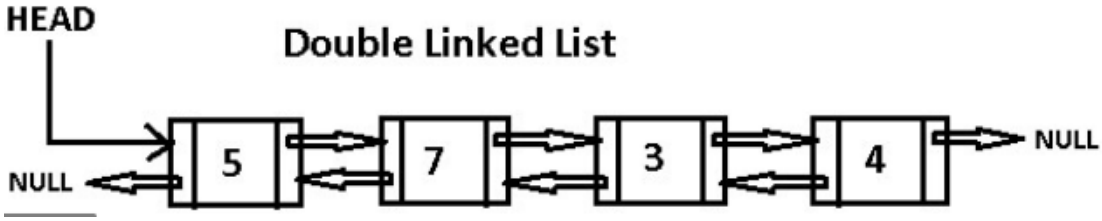
Given the structure of storage of data in SIMLIB, we decided to use the object oriented approach in Python. We made three key computational decisions in Python - 1) Use of Linkedlist class as data structure, 2) Storing SIMLIB lists as array (list in python) of Linkedlist and 3) defining SIMLIB as a class so that all variables remain global. Please note that the User does not have to know these aspects. We discuss these decisions below:

1) Data Structure as Linkedlist in Python

The heart of the SIMLIB is implementation of doubly linked list. This helps in dynamic memory allocation. In C++/C, pointers can be used to implement this structure. However, in Python we created a Linkedlist class with various methods/functions.

Hence, every list can be defined as a Linkedlist class and various methods/functions can be used to operate on these lists. Following methods/Linkedlist functions were implemented in constructing the Linkedlist.

Figure: Diagrammatic representation of Linkedlist



Source: Medium

Node: Each data point with pointer to previous and next object is called Node.

Initiating Linkedlist object: mylist = Linkedlist() . mylist is now a Linkedlist object

Methods created:

- insert_at(data,index): A node with 'data' that is passed will be inserted into the 'index' number.
- remove_at(index): The node from the 'index' will be removed
- copy_from(index): A copy of data from the 'index' will be returned
- get_length(): Gives the length of the linkedlist
- print(): Prints the data in the sequential manner

2) Storing SIMLIB lists as 'List of Linkedlist' in python:

We have defined 'master' as a list of 26 elements. Each element of this list is a Linkedlist object.

3) SIMLIB library is defined as a class:

Implementing global variables in C/C++ is easier as the pointers can help pass the variables by reference. In Python, to alter a global variable in a function required defining then as Global in each function. Hence, we decided to make SIMLIB library a class. All the functions of the SIMLIB are now methods of the object SIMLIB.

III) User Guide

The user does not have to know the computational aspects discussed above. They just need to know a quick snapshot of the functions in SIMLIB and steps to write the main program.

1. Snapshot key variables and constants:

VARIABLES	DESCRIPTION
sim_time	Simulation clock, updated by **simlibPY**
next_event_type	Type of next event; determined by **simlibPY**
transfer[i]	Float array indexed on **i** = 1, 2, …, 10 for transferring attributes of records into and out of lists.
maxatr	max number of attributes in any list; defaults to 10, but user can initialize to < 10 for improved efficiency (cannot be set to < 4 due to the way **simlibPY** works)
list_size[list]	Current number of records in list **list**; maintained by **simlibPY.**
list_rank[list]	Attribute number (if any) on which list is to be ranked (incr. or decr); must be initialized by user.

CONSTANTS	DESCRIPTION
FIRST	Option of filling or removing a record at the beginning of a list, set to 1 in __init__ function.
LAST	Option of filling or removing a record at the end of a list Set to 2 in __init__ function.
INCREASING	Option of keeping a list ranked in increasing order according to the attribute specified in the list_rank array. Set to 3 in __init__ function.
DECREASING	Option of keeping a list ranked in decreasing order according to the attribute specified in the list_rank array. Set to 4 in __init__ function.
LIST_EVENT	Number of the event list, automatically set to 25.
EVENT_TIME	Attribute number of the event time in the event list, automatically set to 1 in __init__ function.
EVENT_TYPE	Attribute number of the event type in the event list, automatically set to 2 in __init__ function.

2. Snapshot key functions:

Functions are of three types – Event/timing, random generator functions and report functions

EVENT/TIMING ROUTINE FUNCTIONS

FUNCTION	INPUT/OUTPUT	DESCRIPTION
__init__		Invoke at beginning of each simulation run. Main function to allocate storage for lists, Initialize all pointer, Set clock to 0, Sets event list for proper ranking on event time, Defaults **maxatr** to 10. Sets all statistical counters to 0.
list_file	I/P -> option : FIRST, LAST,INC,DEC list	File a record in list, according to **option**.
list_remove	I/P -> option : (FIRST or LAST), list	Remove a record from list and copy its attributes into the **transfer** array, according to **option.**
timing		Remove the first record from the event list (the next event), Advance the clock to the time of the next event, Set next_event_type to its type.
event_schedule	I/P -> time_of_event, type_of_event	Invoke to schedule an event at the indicated time of the indicated type. If attributes beyond 1 and 2 are used in the event list their values must be pre-set in the **transfer** array.
event_cancel	I/P -> event_type	Removes the first event of type from the event list, leaving its attributes in transfer.

REPORT ROUTINE FUNCTIONS

FUNCTION	INPUT/OUTPUT	DESCRIPTION
sampst	I/P -> value, variable	Initialize, update, or report statistics on discrete-time processes: sum/average, max (default -1E30), min (default 1E30), number of observations for sampst "variable", where "variable": = 0 initializes accumulators > 0 updates sum, count, min, and max accumulators with new observation < 0 reports stats on variable "variable" and returns them in transfer: [1] = average of observations [2] = number of observations

		[3] = maximum of observations [4] = minimum of observations
timest	I/P -> value, variable	Initialize, update, or report statistics on continuous-time processes:integral/average, max (default -1E30), min (default 1E30), for timest "variable", where "variable": is same as **sampst**
filest	I/P -> list	Report statistics on the length of list "list" in transfer: [1] = time-average of list length updated to the time of this call [2] = maximum length list has attained [3] = minimum length list has attained This uses **timest** variable TIM_VAR + list.
out_sampst	I/P -> unit, lowvar, highvar	Write to file **unit** summary statistics (mean, number of values, max, min) on **sampst** variables **lowvar** through **highvar.**
out_timest	I/P -> unit, lowvar, highvar	Write to file **unit** summary statistics (mean, number of values, max, min) on **timest** variables **lowvar** through **highvar.**
out_filest	I/P -> unit, lowfile, highfile	Write to file **unit** summary statistics (mean, number of values, max, min) on number of records in lists **lowfile** through **highfile**

RANDOM GENERATION ROUTINE FUNCTIONS

FUNCTION	INPUT/OUTPUT	DESCRIPTION
expon	I/P -> mean, stream	Returns in its name a variate from an exponential distribution with mean **mean**, using random-number "stream" **stream**
random_integer	I/P -> prob_distrib[], stream	Returns a variate from a discrete probability distribution with *cumulative* distribution in the array **prob_distrib**, using stream **stream**
uniform	I/P -> a, b, stream	Returns a variate from a continuous uniform distribution on [**a**, **b**], using stream **stream.**
erlang	I/P -> m, mean, stream	Returns a variate from an **m**-Erlang distribution with mean **mean**, using stream **stream.**
lcgrand	I/P -> stream	Random-number generator, returns a variate from the (continuous) U(0, 1) distribution, using stream **stream**.
lcgrandst	I/P -> zset, stream	Sets the random-number "seed" for stream **stream** to **zset.**
lcgrandgt	I/P -> stream	Returns the current underlying integer for stream **stream.**

3. User guidelines to write the main program in SIMLIB

A) Scope of user:

- Still up to user to determine events, write main function and event functions (and maybe other functions), but simlib functions makes this easier
- Determine what lists are needed, what their attributes are
- Determine sampst, timest variables
- Determine and assign usage of random-number streams
- Variables take the place of many state, accumulator variables, but user may still need to declare some global or local variables

B) Typical activities main

- Read/write input parameters
- Invoke init_simlib to initialize simlib's variables
- (Maybe) Set list_rank[list] to attribute number for ranked lists. Also (Speed option) Set maxatr to max number of attributes per list
- (Maybe) timest to initialize any timest variables to nonzero values
- Initialize event list via event_schedule for each event scheduled at time 0
- Events not initially to be scheduled are just left out of the event list
- Invoke timing to determine next_event_type and advance clock
- Invoke appropriate event function, perhaps with case statement
- At end of simulation, invoke user-written report generator that usually uses sampst, timest, filest, out_sampst, out_timest, out_filest

C) Things to do in your program:

- Maintain lists via list_file, list_remove, together with transfer to communicate attribute data to and from lists
- Gather statistics via sampst and timest and Update event list via event_schedule

D) Error checking: We cannot check for all kinds of errors, but some opportunities to do some things in simulation "software" – so simlib checks/traps for:

- Time reversal (scheduling an event to happen in the past)
- Illegal list numbers, variable numbers
- Trying to remove a record from an empty list

IV) Tutorial – Implementing SSQ Model

We just provide a broad overview here. For details, please check the code. The single server queuing model has **two user defined functions – Arrive & Depart**:

- **Arrive function** schedules next arrival, checks if server is busy & stores time of arrival of arriving customer at end of list LIST_QUEUE.
- **Depart function** checks if queue is empty, marks service completion, removes event from the list and store various statistics.

Main function works as follows:

- Initializes the model (were attributes required for model and simlib function is initialized)
- Run the simulation for number of delays specified in input file
- Determine the next event & invoke appropriate arrival or departure event function.
- Lastly, invoke the report generator and end the simulation

```
if __name__ == "__main__":

    # Open input and output files.
    infile  = read_csv('mm1smlb_in.csv')
    mean_interarrival = infile.iloc[0,0]
    mean_service = infile.iloc[0,1]
    num_delays_required = infile.iloc[0,2]

    outfile = open('mm1smlb_out.csv', 'w+', newline ='')
    # writing the data into the file
    with outfile:
        write = csv.writer(outfile)
        write.writerow(list(infile.columns))

    # Set maxatr = max(maximum number of attributes per record, 4)
    maxatr = 4

    # Initialize the model
    ssq = singleServerQueueing(mean_interarrival)

    # Run the simulation while more delays are still needed.
    while (ssq.num_custs_delayed < num_delays_required):
        Determine the next event.
        ssq.spy.timing()

        # Invoke the appropriate event function.
```

```
        if (ssq.spy.next_event_type == ssq.EVENT_ARRIVAL):
            ssq.arrive(mean_service)
            break
        elif (ssq.spy.next_event_type == ssq.EVENT_DEPARTURE):
            ssq.depart(mean_service)
            break
        else:
            print("error")


    # Invoke the report generator and end the simulation.
    ssq.report()
```

OUTPUT FILE

```
Single-server queueing system using simlib
Mean interarrival time      1.000 minutes
Mean service time           0.500 minutes
Number of customers         1000
Delays in queue, in minutes:
 sampst                     Number
variable                     of
 number      Average         values        Maximum        Minimum
_____________________________________________________________________________
   1       0.524873         1000.00        5.63309        0.000000
_____________________________________________________________________________
Queue length (1) and server utilization (2):
  File       Time
 number      average         Maximum        Minimum
_____________________________________________________________
   1       0.540077         8.00000        0.000000
   2       0.510693         1.00000        0.000000
_____________________________________________________________
Time simulation ended:     971.847 minutes
```

Conclusions and Learning

Implementing SIMLIB improved our understanding of how event-driven discrete software function at the most basic level. We learned how the different routines co-ordinate to create a nice simulation. We have already discussed our understanding of how SIMLIB works in the previous section. Also, we realized that there advantages and a few drawbacks of using SIMLIB. Our next step can be make use parallel & distributed programming concepts to make efficient even for complex applications

Using SIMLIB has advantages and drawbacks:

Advantages:

- Framework: SIMLIB provides a good framework with most of the important components that a programmer can use in programming a simulation
- SIMLIB uses Linkedlist allocation which has various advantages like easy adding, deleting, inserting of records and reduces memory requirements as compared to sequential allocation.
- Object Oriented approach possible: Since SIMLIB can be coded in Python, Java, etc, an object oriented approach can be used which is preferred by most programmers.
- Cost effective: The cost of animated driven/user friendly software is high while SIMLIB is open source and hence free
- Efficient & tailored: SIMLIB can be used to simulate tailor made application and hence can be more efficient compared to generic software that is designed for a range of applications.

Drawbacks:

- More time required to program the simulation: SIMLIB only provides basic functions; the programmer has to create most of the features in a complex simulation.
- Error detection is time consuming: An error may be very difficult to find as it propagates through the various functions.
- Modifying the model is time consuming: Sometimes minor modifications in the model may require the programmer to make many changes and think through the entire code.
- No animation and difficult to learn: Compared to other process interaction software, SIMLIB has no animation and hence it is more difficult to learn. Hence higher training cost

What next?

More features: Some more routines for events can be added based on the type of applications we want to use the SIMLIB.

Parallel and distributed Simulations: Once more features are added, we can focus on using recent advances in computing especially, distributed computing and parallel computing. When simulating complex systems, the concepts of parallel computing can be used for making the simulations faster. Moreover, distributed computing can be used to run multiple simulations and connect all of them into a co-ordinate complete simulation.

Reference:

- **Book - *Simulation modeling and analysis by Averill Law***
- ***ISYE 6644 Simulation lectures***