gpss QUEUE stats in the .rpt file CSC148, 6 Oct 2020

This document demos the stats automatically generated for a QUEUE/DEPART block pair.

Histograms, as produced using TABLE statements, and savevalue entities are 2 additional ways to create stats, but histograms and savevalues require additional user-written code.

This demo shows that all existing savevalue entities and their values are at the end of each .rpt.

**Source code**

; gpssW demo for illustrating .rpt file QUEUE and user-specified stats

; Use RN1's default seeding (thus, does not need an RMULT block)

clear

iaMean EQU 0.5 ; {iaj} mean - one cj, an average, every 2 t.u.

serviceDurMean EQU 0.25 ; avg(service duration)

initial x$cjNumber,0 ; Incremented after each cj arrival

generate (exponential(1,0,iaMean)) ; cj interarrrivals are exponentially distr.

savevalue cjNumber+,1 ; Increment cj counter upon each cj arrival

queue res\_dur ; Start stats collection for cj residence duration

queue entryDel ; Start stats collection for cj activities at entry

advance (RN1@2#0.5) ; cj does unspecfied actions, in 0 or 0.5 t.u.

depart entryDel ; Finish stats collection for cj activities at entry

seize svr ; Start cj service, else wait until server is free

advance (exponential(1,0,serviceDurMean)) ; cj gets service at svr

release svr ; Free up the server

depart res\_dur ; Finish stats collection for cj residence duration

terminate 1 ; cj leaves the system

**Report results**

GPSS World Simulation Report - gpss\_QUEUE\_stats\_demo.8.1

Tuesday, October 06, 2020 17:40:57

START TIME END TIME BLOCKS FACILITIES STORAGES

0.000 4987.370 11 1 0

NAME VALUE

CJNUMBER 10002.000

ENTRYDEL 10004.000

IAMEAN 0.500

RES\_DUR 10003.000

SERVICEDURMEAN 0.250

SVR 10005.000

LABEL LOC BLOCK TYPE ENTRY COUNT CURRENT COUNT RETRY

1 GENERATE 10001 0 0

2 SAVEVALUE 10001 0 0

3 QUEUE 10001 0 0

4 QUEUE 10001 0 0

5 ADVANCE 10001 0 0

6 DEPART 10001 0 0

7 SEIZE 10001 1 0

8 ADVANCE 10000 0 0

9 RELEASE 10000 0 0

10 DEPART 10000 0 0

11 TERMINATE 10000 0 0

FACILITY ENTRIES UTIL. AVE. TIME AVAIL. OWNER PEND INTER RETRY DELAY

SVR 10001 0.504 0.252 1 10000 0 0 0 0

QUEUE MAX CONT. ENTRY ENTRY(0) AVE.CONT. AVE.TIME AVE.(-0) RETRY

RES\_DUR 13 1 10001 0 1.503 0.750 0.750 0

ENTRYDEL 5 0 10001 4915 0.510 0.254 0.500 0

SAVEVALUE RETRY VALUE

CJNUMBER 0 10001.000

CEC XN PRI M1 ASSEM CURRENT NEXT PARAMETER VALUE

10000 0 4986.621 10000 7 8

FEC XN PRI BDT ASSEM CURRENT NEXT PARAMETER VALUE

10002 0 4987.692 10002 0 1

**.rpt QUEUE stats**

Each QUEUE/DEPART block pair has a corresponding line of simple stats in the .rpt file.

MAX, AVE.CONT., and AVE.TIME are the most-used of the info on each such line.

The QUEUE/DEPART pair’s A operand named RES\_DUR gathers stats for cj residence time, and the QUEUE/DEPART pair with A operand named ENTRYDEL gathers (trivial) stats for the brief timeframe when the cj are doing the activities at model entry.

RES\_DUR stats: MAX 13 means 13 was the largest number of cj in S at any time in the run;

1.503 ( AVE.CONT. ) was the average number of cj in S, aka “L”, and .750 ( AVE.TIME) was the average time spent in S (i.e. average residence time), aka W, by the cj.

Question: Why is AVE.CONT. twice as large as AVE.TIME ?

First level of Answer: the relationship of these values depends on the code region of the corresponding QUEUE/DEPART blocks. For the RES\_DUR report line, the region is the whole system, corresponding to the entire time each cj is in S.

Detailed Answer: By Little’s Law, L = l\*w and since iaMean (in the source code) is 0.5, l is thus 1/.5 = 2, so L = 2 \* .750 = 1.5 (The stat in the model is just that, a stat, that should be close to AVE.CONT. but even for **Start 10000**, it is not exactly the value predicted by Little’s Law, but it is close, as it should be.

(BTW, you should confirm that the model is stable (as needed to apply Little’s Law))

The exact same analysis applies to the .rpt QUEUE line for ENTRYDEL and explains why AVE.CONT. is twice as large as AVE.TIME. In both cases, the arrival process is the same distribution.

The savevalue entity is a simple illustration that more user-written code is needed to obtain other stats beyond what .rpt provides with no coding at all, as with the QUEUE .rpt section. Look at the savevalue info near the end of the .rpt and the source code involving savevalue cjNumber

If a tr exists at the end of a model run, then the .rpt also displays all existing parameters for each such tr. (This simple model had no tr parameters)

**Histograms**

Histograms mainly provide distribution info about random variables of interest in a model.

But, they need parameters (4 TABLE operand values) for defining histogram display properties.

**Follow ups for coming classes**

A histogram derived from data collection D about a system S is one of 2 common ways that a DES development team can determine an appropriate theoretical distribution D (uniform/Poisson/Pareto/ … ?) to use for modeling the info in D. The translation from D to D is the topic of h3.

gpss uses the very simple TABLE statement and 4 operands, all required to render a histogram, and a corresponding TABULATE block to get the current value of the random variable indicated by the A operand.

Debunking some zoom chat I have seen re. “gpssW is a low-level language”

The opposite is true.

The 2 gpssW statements to render a histogram compare with approx. 10 LOC to do the same with the python pyplot utility or the approx.. 20\_ LOC needed using the Java Histogram class.

Debunking some zoom chat I have seen re. TABLE:

There is NO REQUIRED USE of a QUEUE/DEPART block pair in order to generate a histogram => Witness the histogram the above source code produces. The 10000 samples of an int in [0,9] are not exactly uniform in the histo below, but do confirm that the gpssW “@” operator works, as advertised.

