Fiabilitat d'un algorisme generador de nombres aleatoris en diferents llenguatges de programació

Jan Pàmies, Aniol Carbó, Ivan Lagunas

Novembre 2019

Introducció

En l'àmbit de la informàtica, el disseny d'algoritmes fiables i eficients és molt rellevant. És per això que volem estudiar si, dins l'aleatorietat d'un algorisme que selecciona un nombre 'random' d'un interval, podem afirmar que un és més fiable que un altre depenent en quin llenguatge de programació està escrit. Els llenguatges que utilitzarem seran C++ i JAVA.

Material y mètodes

Hem preparat 2 programes, que generen un nombre aleatori, en els 2 llenguatges de programació escollits. Utilitzarem el mateix ordinador per a fer les proves, d'aquesta manera no es veuran afectats els resultats per possibles factors externs al codi.

Executarem 20 vegades cada algorisme, deixant un temps entre el canvi de codis per evitar problemes o interferències de resultats.

Després, realitzarem els càlculs corresponents per a poder afirmar quin és l'algorisme més fiable dels 2.

Disposem de les variables explicatives:

1. Math.random() (de Java)
2. rand (de C++)

Les quals defineixen l'algorisme utilitzat. Els valors obtinguts amb l'execució dels algorismes ens ajudaran a obtenir la variable resposta, encarregada de decidir quin és el millor algorisme en termes de qualitat.

Un dels criteris que es poden utilitzar per a valorar una funció generadora de nombres aleatoris és el següent:

El codi genera 1000 nombres aleatoris, que poden anar del 0 al 99. D'aquesta manera obtenim que el valor esperat és de 10 cada nombre (1000/100). Un cop trobat el nombre de vegades que obtenim cada número, hem de restar-ho (elevant-ho al quadrat) per saber a quina distància estem del valor esperat. Fent el sumatori dels resultats per a tots els valors de 0 a 99 obtenim la dispersió total de l'algorisme respecte el valor esperat.

índex de dispersió = $\sum_{i=0}^{99}$ ((Ni-10)^2)

Executarem els dos programes 20 vegades cadascun i crearem un gràfic Box-Plot i un gràfic Normal Q-Q Plot amb les dades obtingudes en totes les execucions.

I. Premisses convenients

1. Normalitat dels valors de la variable resposta per els dos algorismes
2. Homoscedasticitat: variància constant entre els algorismes
3. Independència de les dades obtingudes: escollides a l'atzar a través de l'algorisme

II. Contrast d'hipòtesi:

H_0: $\mu_{C++} = \mu_{JAVA}$

$H_1 = \exists_{i,j}$ tq $\mu_i \neq \mu_j$

Objectius

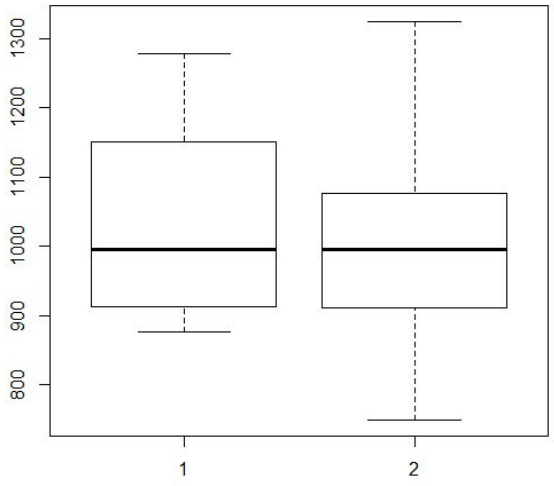
La nostra recollida de dades es basa en obtenir la dispersió del nombre de vegades que es genera aleatòriament cada número (rang de 0 a 99). Generant doncs, 1000 nombres, tenim que el valor esperat és 10.

A l'hora de decidir com interpretem aquestes dades ens hem adonat de que no es pot definir un criteri que digui quin dels dos algorismes és millor. El motiu és el següent: dins l'aleatorietat de la generació d'un nombre, si diem que un algorisme és millor perquè genera cadascun dels nombres X vegades (és a dir, de manera equilibrada), el concepte d'aleatorietat desapareix, ja que es podria predir durant l'execució del programa quin nombre es generaria basant-nos en les probabilitats que té de sortir mirant els que ja hem obtingut. És per això que hem decidit basar-nos en aquesta reflexió dient que l'algorisme que tingui un "índex de dispersió" més baix, serà l'algorisme més **determinista**.

Resultats

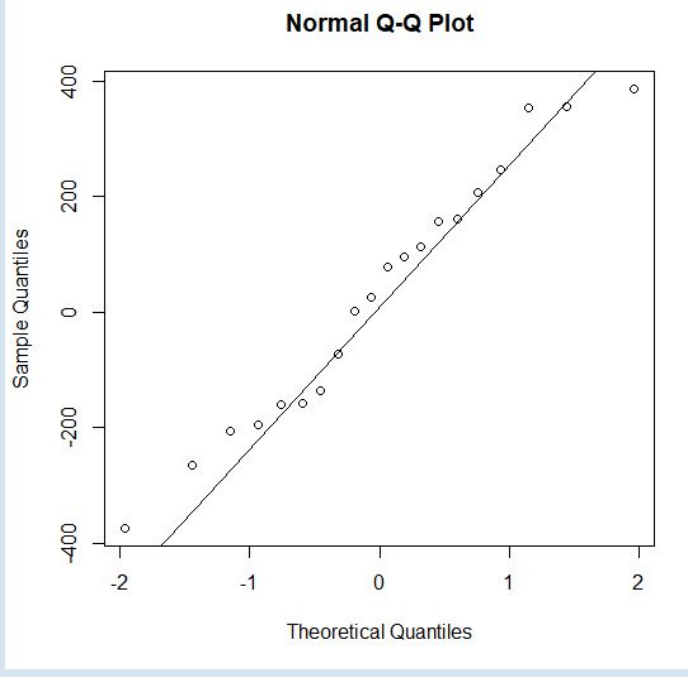
Llenguatge de programació	Mitjana	Desviació estàndard	IC95
JAVA	1028.6	131.5464	[969.0531 , 1088.1469]
C++	998	143.8142	[932.8999 , 1063.1001]

(recull simplificat de les dades en aquesta taula, el recull complet al document adjunt)



Gràfic 1: Box-plot. El de l'esquerra és el de Java i el de la dreta el de C++

Podem observar que tenen una mitjana molt semblant, però els casos aïllats són més dispersos en el cas de C++ (bigotis més allargats). Tot i així, és també el cas en què l'interval entre el primer i el tercer percentil és més petit. Contràriament, d'aquesta manera obtenim que en cas de Java la caixa és més ampla, però amb uns casos aïllats més propers.



Gràfic 2: Veiem que els punts s'ajusten bastant bé a la recta. Per tant, podem assumir normalitat al veure que s'ajusta bastant als quantils teòrics de la Normal i es manté la hipòtesi nul·la.

Equació de regressió i coeficient de determinació

L'equació de regressió de JAVA sobre C++ és:
valor_JAVA = 1293.1703 - 0.2651 * valor_C++

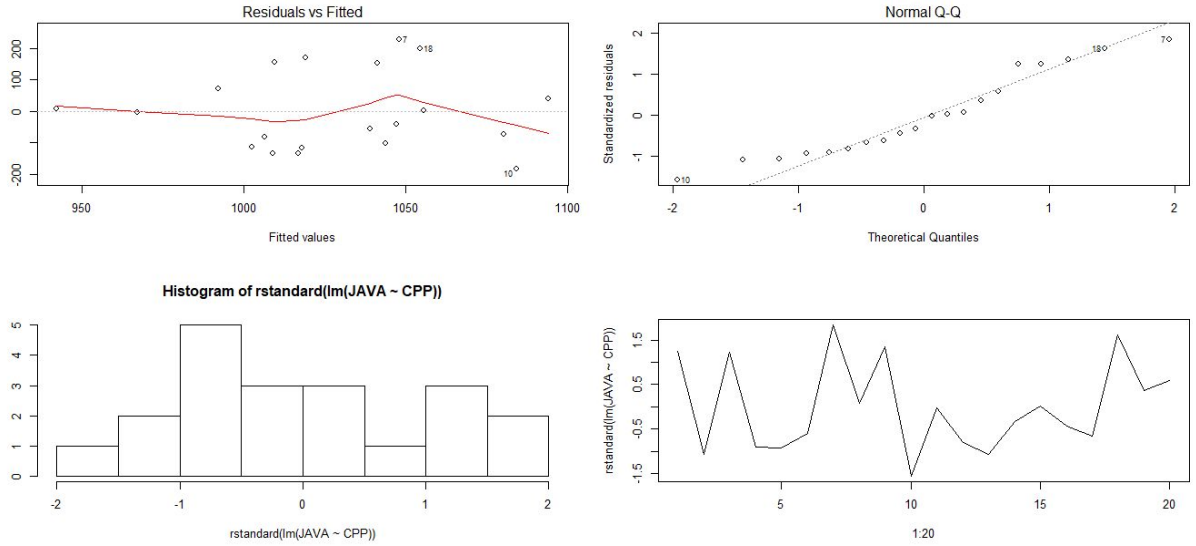
L'equació de regressió de C++ sobre JAVA és:
valor_C++ = 1323.9136 - 0.3169 * valor_JAVA

El coeficient de determinació (R-squared) val:

$R^2 = 0.084$

Com que el coeficient de determinació és molt baix vol dir que hi ha una gran variabilitat d'origen aleatori, és a dir, que Y no s'explica pel factor X ni X s'explica pel factor Y.

Anàlisi de les premises



Analitzant el primer gràfic, veiem que el núvol de punts segueix una alçada bastant constant, de manera que podem dir que tenim linealitat. A més, la variabilitat dels residus es manté constant independentment dels valors predits, de manera que tenim també homoscedasticitat.

Amb el segon gràfic afirmem la normalitat dels residus ja que estan situats (majoritàriament) sobre la recta. Tot i que els resultats presenten una desviació a l'alçada de l'1, tornen a recuperar la seva posició sobre la recta, de manera que podem afirmar normalitat tot i aquest entrebanc. A més, tenim l'histograma (tercer gràfic) per analitzar la

normalitat dels residus. Tot i que el Q-Q Norm és més fiable que l'histograma, el segon ens permet confirmar la normalitat de residus que podem dubtar del primer per la lleugera desviació que presenta.

A través del quart gràfic podem concloure independència dels resultats ja que els residus no segueixen cap patró previsible.

Encara que són poques dades, res s'oposa a validar cap de les 4 premises.

P-valor

El p-valor associat és 0.9443922, i com que és superior al risc previ ($\alpha = 0.05$) no podem refutar l'hipòtesi nul·la:

$$H0: \mu c++ = \mu JAVA$$

No podem afirmar doncs que un algorisme sigui més determinista que l'altre, però hem pogut observar que en en el cas de JAVA la seva dispersió és menor i, per tant, té una tendència lleugerament més determinista que l'algorisme de C++.

Limitacions

No podem afirmar que aquestes conclusions es compleixin en tots els algorismes generadors de nombres aleatoris ja que només n'hem seleccionat 2 per a fer aquest estudi. A més, el nombre de dades generades potser no és l'adequat depenent de la situació en què es volen utilitzar els algorismes i les dades que es necessiten.
Així doncs, per als possibles estudis futurs s'haurien d'agafar més dades i incloure altres algorismes a l'estudi.