公交车悖论

June 2025

摘要

在生活中,我们经常会发现,公交车比自己想象中难等。本篇文章针对等公交车这一生活现象进行研究,分别从数学理论推导,计算机程序模拟,心理学因素等多方面进行论述,解释难等公交车的主要原因。在此基础上,解释检查悖论这一存在于公交车悖论背后的原理。

关键词: 等待,公交车悖论,检查悖论,期望,模拟。

1 引言

检查悖论,是指当观察量的概率与观察量有关时出现的一种现象。具体来说,检查悖论描述的是一种情况,即观察者在进行统计或测量时,由于观察行为本身对被观察对象产生了影响,从而导致观察结果与实际情况不符的现象。比如一个大学里有人多的大班,也有人少的小班,你想调研一下班级的平均人数,就随机问了 50 个同学班级有多少人再取平均。你觉得你的抽样很随机,这个结果似乎是正确的,若统计样本足够多,最后的结果可能是所有班级的平均人数。但事实并非如此,因为大班的人数多,在抽样过程中,这 50 个人更可能来自大班,而小班中的学生则相对不容易调查到,你的调研结果一定偏大。如果你是随机抽样测平均身高,那其实没问题,但正因为你是在人数中抽样来调研班级的平均人数,那你自以为的随机抽样在无形中就更容易抽到人数更多的大班。

在观看了网上的视频后,检查悖论导致的另一生活现象引起了我们的注意: 公交车悖论。假设你要坐公交车去一个地方,你到了公交车站,要乘坐的公交车还没到,那么你的等待时间将会是多久呢?

假设你要乘坐的公交车每小时有 10 个班次,那么平均每 6 分钟就会来一班次。如果最近的一班车在你到达之前刚好离开,你可能要等 5-6 分钟; 如果最近一班车在你到站之前已经离开很久了,那么下一班车也许很快就来,因此,你的预期平均等待时间是 3 分钟。然而,这种直觉性的预期大多数都不正确,你的实际平均等待时间往往超过了 3 分钟,甚至可能比 6 分钟还要长。这种现象被称为“公交车悖论”。

本文围绕公交车悖论这一核心命题,先从数学上直观证实其正确性,接着用计算机程序模拟不同路况下等待时间的结果,再从心理学因素分析这一问题,最后从检查悖论这一宏观的问题入手,总结全文。

2 数学推导

首先,我们从数学角度证实公交车悖论的正确性,通过两种不同的生活情形计算等待的时间期望。

Theorem 2.1. *若公交车的到达时刻表已知,则在该段时间内,人所需要等公交车的时间期望 $E = \frac{1}{2}(\frac{\sigma^2}{\bar{n}} + \bar{n})$,其中 σ^2 为相邻公交车到达公交站的时间差的方差,$\bar{n}$ 为相邻公交车到达公交站的时间差的平均值。*

Proof of Theorem 2.1. 设我们从 0 时刻开始研究该问题,第一辆公交车经过 n_1 的时间到达站台,在第 i 辆公交车到达公交站后,再经过 n_{i+1} 的时间才到达公交站台,且一共有 t 辆公交车经过站台。在第 t 辆公交车到达站台后,停止考虑该问题。记 $m = \sum_{i=1}^{t} n_i$。则:

$$E = \sum_{i=1}^{t} \frac{n_i}{m} \cdot \frac{n_i}{2} = \frac{1}{2m} \sum_{i=1}^{t} n_i^2, \tag{1}$$

且

$$\sigma^2 = \frac{1}{t} \sum_{i=1}^{t} (n_i - \frac{m}{t})^2 = \frac{1}{t} (\sum_{i=1}^{t} n_i^2 - \frac{2m}{t} \sum_{i=1}^{t} n_i + \frac{m^2}{t}), \tag{2}$$

$$t\sigma^2 = \sum_{i=1}^{t} n_i^2 - \frac{m^2}{t}, \tag{3}$$

故代入化简有

$$E = \frac{1}{2m}(t\sigma^2 + \frac{m}{t^2})) = \frac{1}{2}(\frac{\sigma^2}{\bar{n}} + \bar{n}). \tag{4}$$

□

Theorem 2.2. *若在某 0 时刻后,总计时间 T 内共随机到达了 n 辆车。已知一个人在 t 时刻开始等车,人所需要等公交车的时间期望 $E(t) = \frac{t^{n+1}+T^{n+1}}{T^n(n+1)}$*

Proof of Theorem 2.2. 考虑某段时间微元 $t+x$ 与 $t+x+dx$,在该区间等到车的概率为:

$$P(x) = n \cdot \frac{dx}{T} \cdot (\frac{T-x}{T})^{n-1}. \tag{5}$$

等不到车的概率为: $(\frac{t}{T})^n$,于是:

$$\begin{aligned} E(t) &= \int_0^{T-t} n \cdot \frac{dx}{T} \cdot (\frac{T-x}{T})^{n-1} + (\frac{t}{T})^n \\ &= \frac{n}{T^n} \int_0^{T-t} x(T-x)^{n-1} dx + \frac{t^n}{T^{n-1}} \\ &= \frac{t^{n+1} + T^{n+1}}{T^n(n+1)} \quad . \end{aligned} \tag{6}$$

特别的当 $t=0$ 时有:

$$E(0) = \frac{T}{n+1}. \tag{7}$$

□

Remark

*Remark*1 : 结论 3.1 与 3.2 在实际生活中的应用各有利弊。结论 3.1 需要已知公交车到达的时刻表, 条件苛刻, 更多是在公交车已经全部运行完事后对人等公交车的时间期望的计算结果。而结论 3.2 是提前预测之后的结果, 但是需要在公交车随机到达的条件下进行研究,在实际情况下会不够准确。

*Remark*2 : 结论 3.1 中,我们可以直观地发现:

(1) 由于地铁相邻两班的到达时间差为固定的时间 t,则乘坐地铁等待的时间期望为 $\frac{t}{2}$。

(2) 由均值不等式:

$$E = \frac{1}{2}(\frac{\sigma^2}{\bar{n}} + \bar{n}) \tag{8}$$

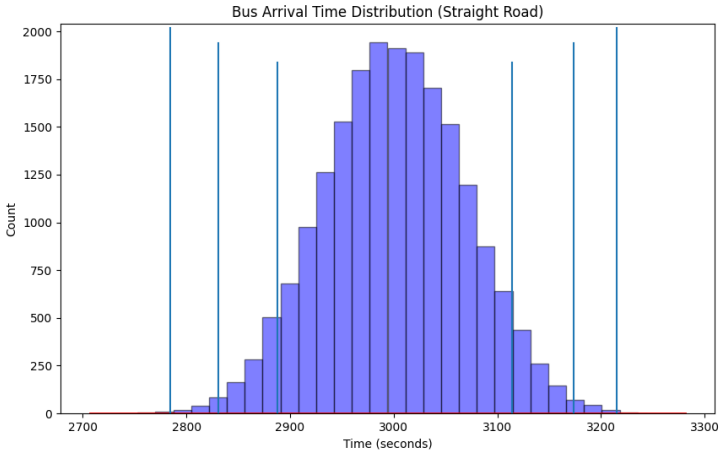
满足 $E \geq \frac{1}{2}\bar{n}$ 且 $E \geq \sigma$。事实上,(1) 中的情形即为期望最小的情况。

3 计算机模拟

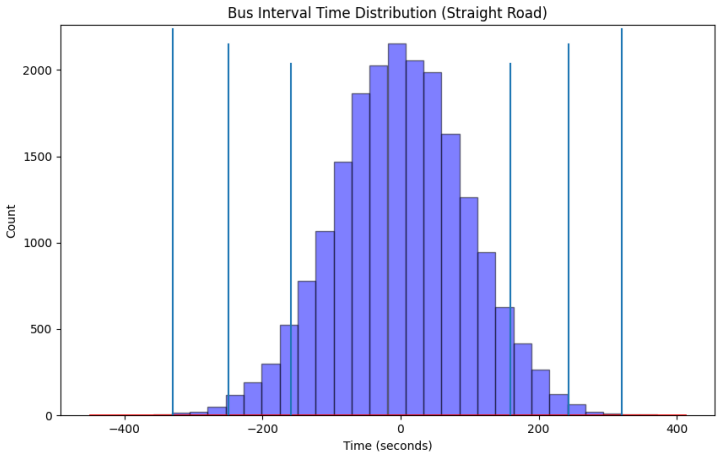
针对数学推导的结果,我们使用 Python 进行编程,对公交车的路况进行模拟,探究不同路况对等待时间期望的影响。使用大量公交车作为模拟样本,对每个公交车的行驶时间通过已知的概率分布进行模拟,同时构建一个 [0 1] 到时间的映射作为概率分布,得到了以下主程序思路和结论:

在整个程序中,$randomness$ 体现为将原本已知的公交车到站的时间分布进行适当打乱,$randomness$ 越大,实际分布与原函数的分布的相异程度越高,并定义 $randomness$ 值为 k 时,此时映射为 $[0\ 1]$ 到 $[1-k\ 1+k]$ 的映射。在 $randomness$ 取 0 时,严格按照程序输入的车辆行驶概率分布,实际模拟中,统一取混乱度为 0.125。在上述程序中,我们认为两个公交车在始发站相邻车发车时间差为 5 分钟。我们模拟得到三个数据,分别为等待公交车的时间期望, 公交车从始发站开始计时到达该站所需时间分布, 相邻两辆到达该站的时间差与始发站发车时间差的时间差分布。在下面所有分布图中, 图中三组竖线由中间到两侧分别表示该分布的第 90 、第 99 和第 99.9 百分位数,则对于不同路况有以下数据分析:

(I) 畅通无阻的路。(例如: 高速或快速路等)
经过程序模拟,在 0.125 的混乱程度下,等待时间期望为 $165.44s$,分布图如下:



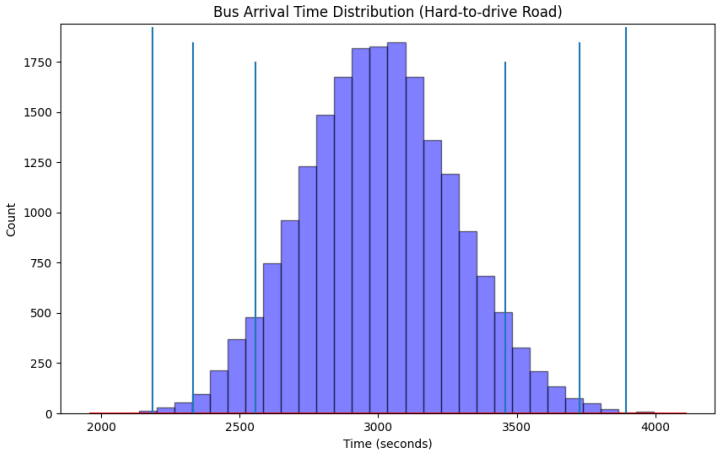
(a) Bus Arrival Time Distribution



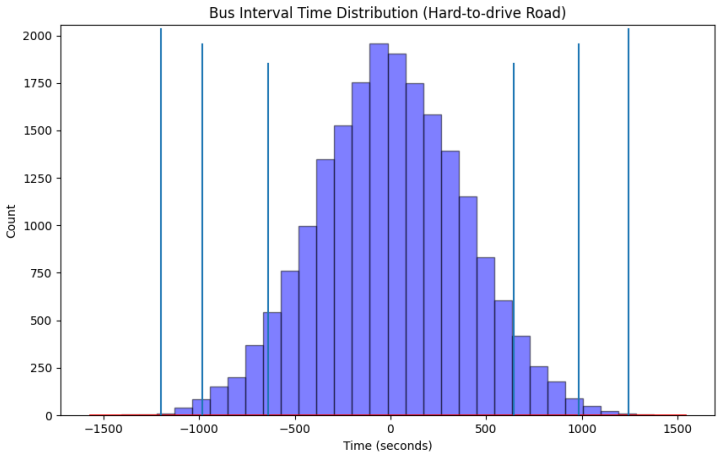
(b) Bus Interval Time Distribution

图 1: Straight

(II) 遇到了路况复杂的路。(例如: 乡间小道,一会堵车一会不堵车等)经过程序模拟,这时候乘客等待的时间期望为 $369.70s$,分布图如下:



(a) Bus Arrival Time Distribution

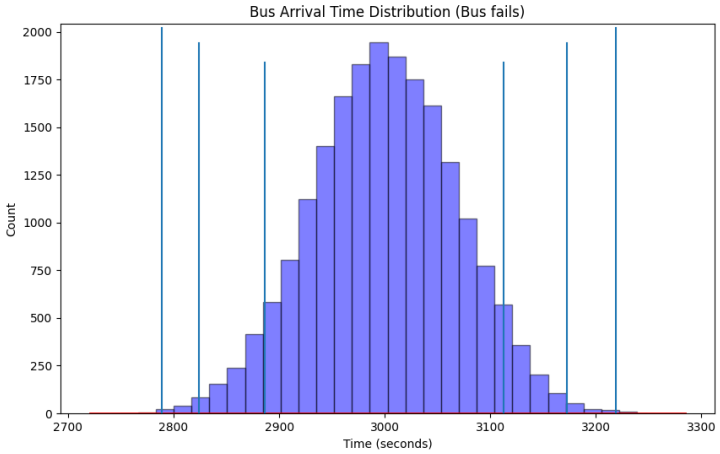


(b) Bus Interval Time Distribution

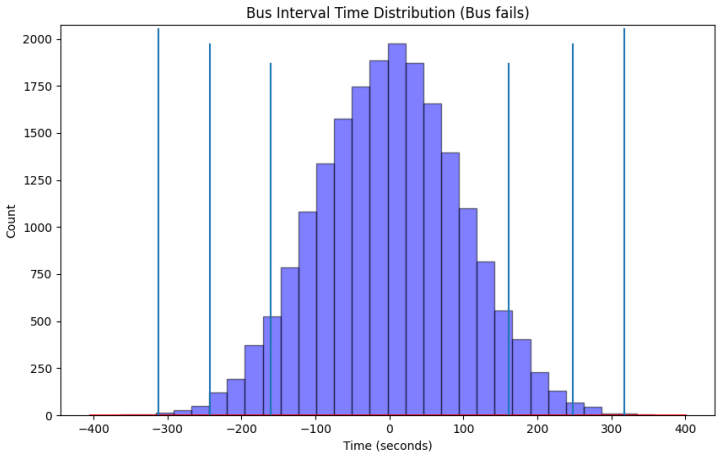
图 2: Hard

(III) 实际情况中,公交车可能会有 1% 的概率在路上出现一些故障。

经过程序模拟,此时乘客等待时间期望为 170.14s,与畅通无阻的路相比,唯一增量来源是故障导致的 (乘客只能等下一班没有故障的车)。



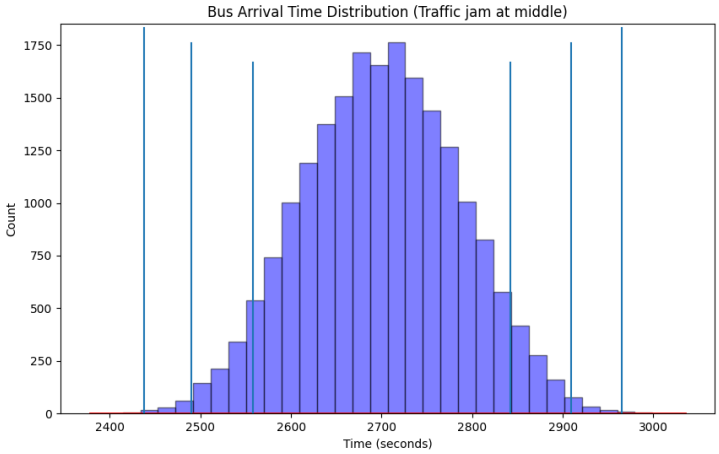
(a) Bus Arrival Time Distribution



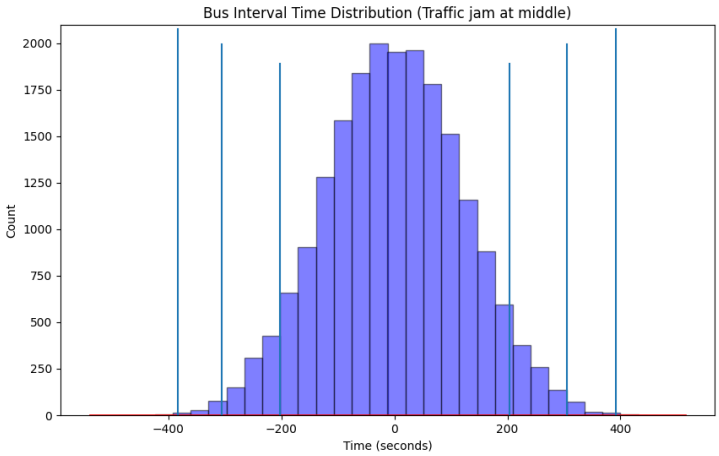
(b) Bus Interval Time Distribution

图 3: Fail

(IV) 公交车在路途中遇到堵塞。我们分成故障在路途的中段和末段进行模拟,得到等待期望均为 175.00s,分布图如下:

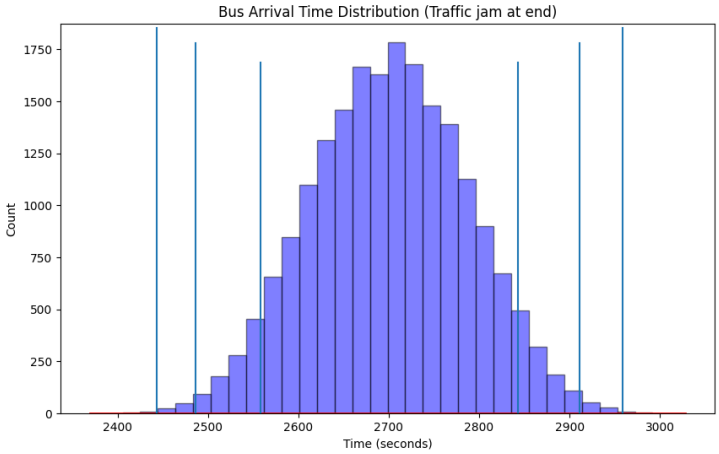


(a) Bus Arrival Time Distribution

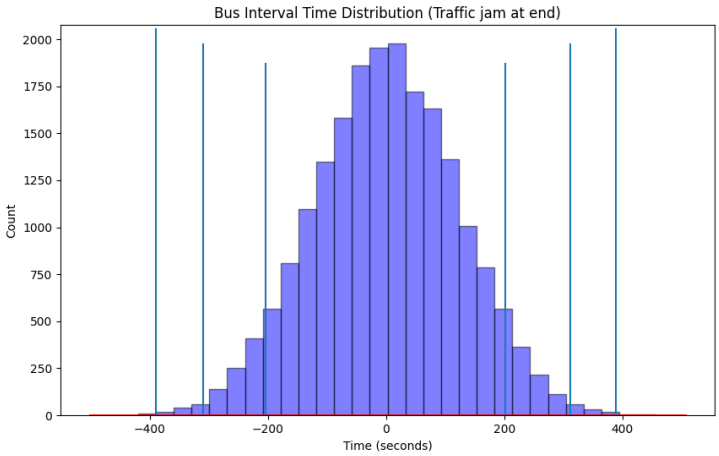


(b) Bus Interval Time Distribution

图 4: Traffic jam in the middle



(a) Bus Arrival Time Distribution



(b) Bus Interval Time Distribution

图 5: Traffic jam at the end

分析:

(1) 从模拟中,路况复杂的路的公交车行驶分布最为杂乱。

(2) 畅通无阻的路到达某一站的公交车所用时间方差小。

(3) 堵塞出现的位置与公交车行驶所需时间无明显联系。

(4) 堵塞情况乘客的等待时间期望似乎比实际情况小, 主要原因是在编写程序时采取的是轻微堵塞进行模拟。

4 心理学因素

等待本身是一件很折磨的事情。在 $New\ York\ Times$ 的文章中提到:
Our expectations further affect how we feel about waiting。Uncertainty magnifies the stress of waiting, while feedback in the form of expected wait times and explanations for delays improves the tenor of the experience.

这告诉我们,当我们等待一个完全不知道什么时候会到达的公交车,会增加等待时期的焦虑,并认为等待时间很长。除此以外,有的时候我们急需公交车,这种预期会影响我们对公交车的等候感觉。

5 检查悖论

检查悖论描述的是在随机时刻"检验"某个事件过程时,观测到的时间间隔往往比实际平均间隔更长的现象。当你任意寻找一段时间区域 $p(t)$ 时,$p_{inspect}(t) = \frac{tp(t)}{E(T)}$,这就意味着较长的时间被抽到的概率会更高。此时,类似的,被检查的期望

$$E = \int_0^\infty t p_{inspect}(t) dt = \frac{E(T^2)}{E(T)} \geq E(T). \tag{9}$$

在公交车的例子中,就可以这样理解: 假如公交车每隔 10 分钟来一趟,则当你随机走上站台时,你更有可能遇到的是时间长的一班公交车,所以你的平均等待时间会更长。

6 总结

本文致力于探索公交车等待时间这一主题问题,我们从数学推导,计算机模拟,心理学因素等多方面进行研究,尤其是对于计算机模拟,我们模拟了不同路况的情况,从简单到复杂,给出了各类结果,对问题有了较为清晰的认识。

7 致谢

衷心感谢王奉超老师对文章的指导。

参考文献

[1] Barry Schwartz. Why Waiting in Line Is Torture. The New York Times, August 2012.

[2] Jake VanderPlas. The Waiting Time Paradox. Jake VanderPlas's Blog, September 2018.

[3] 毕导 THU. [毕导] 看了这个视频,你会释怀你倒霉的一生. Bilibili, 2024. 视频上传日期:2 August 2024.

附录

附上模拟程序。

Listing 1: bus monitoring

```
import random

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import math

# DEFAULT VALUES:
# RANDOMNESS = 0.1
# ITER = 100
# PROJECTEDTIME = 3000
# TOTALBUS = 20000

TIMEDATA = []
INTERVALDATA = []

import random

class randomizer:
    def __init__(self, randomness):
        if isinstance(randomness, float):
            # Convert float to a callable lambda
```

```
            self.randomizer = lambda: (1 + randomness * (-1 + 2 *
                random.random()))
        else:
            # Assume it's already a callable function
            self.randomizer = randomness

    def reload(self, new_randomness):
        if isinstance(new_randomness, float):
            self.randomizer = lambda: (1 + new_randomness * (-1 + 2
                * random.random()))
        else:
            self.randomizer = new_randomness

    def __call__(self):
        return self.randomizer()  # Now safe: randomizer is always
            callable

def flatten(iterable):
    flattened = []
    for item in iterable:
        if isinstance(item, (list, tuple, set)):
            flattened.extend(flatten(item))
        else:
            flattened.append(item)
    return flattened

class randomprocess:
    def __init__(self, rand: list):
        if type(rand) == float: rand = [rand]
        r = flatten(rand)
        self.r = [randomizer(i) if not isinstance(i, randomizer)
            else i for i in r]
    def calc(self, iter = 100, projectedtime = 3000):
        ITER = iter
```

```
        PROJECTEDTIME = projectedtime
        totalTime = 0
        t = []
        t.extend(self.r)
        if len(self.r) == 0: raise Exception("UwU")
        else:
            while len(t) < iter: t.extend(self.r)
        for i in t:
            totalTime += PROJECTEDTIME / ITER * i()
        return totalTime

def SpawnBus(randomness = 0.1, iter = 100, projectedtime = 3000):
    rd = randomprocess(randomness)
    return rd.calc(iter, projectedtime)

def DrawDistribution(data, title, xlabel, ylabel, save, filename):
    plt.figure(figsize=(10, 6))
    counts, bins, _ = plt.hist(data, bins=30, alpha=0.5, color='blue
        ', edgecolor='black', density=False)
    bin_width = np.diff(bins)
    scaled_counts = counts / (sum(counts) * bin_width)
    plt.bar(bins[:-1], scaled_counts, width=bin_width, alpha=0.5,
        color='blue', edgecolor='black')
    sns.kdeplot(data, color='red', linewidth=2)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    data = sorted(data)
    plt.axvline(data[math.floor(len(data) * .95)], 0, 0.9)
    plt.axvline(data[math.floor(len(data) * .05)], 0, 0.9)
    plt.axvline(data[math.floor(len(data) * .995)], 0, 0.95)
    plt.axvline(data[math.floor(len(data) * .005)], 0, 0.95)
    plt.axvline(data[math.floor(len(data) * .9995)], 0, 0.99)
    plt.axvline(data[math.floor(len(data) * .0005)], 0, 0.99)
```

```
86      if save == 1:
87          plt.savefig(filename, dpi=300)
88      plt.show()
89
90  def SimulateBus(randomness = 0.1, iter = 100, projectedtime = 3000,
        totalbus = 20000, extratext = ""):
91      time = []
92      interval = []
93      prev = 0
94      cur = 0
95      for i in range (0, totalbus):
96          prev = cur
97          cur = SpawnBus(randomness, iter, projectedtime)
98          time.append(cur)
99          if i > 0: interval.append(cur - prev)
100     time_descr = 0
101     for i in interval: time_descr += (300 + i) ** 2
102     time_descr /= 2 * len(interval) * 300
103     print(time_descr)
104     DrawDistribution(time, "Bus␣Arrival␣Time␣Distribution␣("+
            extratext+")", "Time␣(seconds)", "Count", 1, "time.png")
105     DrawDistribution(interval, "Bus␣Interval␣Time␣Distribution␣("+
            extratext+")", "Time␣(seconds)", "Count", 1, "interval.png")
106
107 randomness = [0.125, 0.125,
108     #lambda: (math.e - 1) ** -1 * math.e ** random.random(),
109     lambda: (2 * random.random() - 1) ** 1/3,
110     0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125
111 ]
112
113 randomness2 = [lambda: (math.e - 1) ** -1 * math.e ** random.random
        ()]
114
115 def main():
```

```
116     # SimulateBus(randomness = 0.005, extratext = "randomness =
            0.005")
117     # SimulateBus(randomness = 0.010, extratext = "randomness =
            0.010")
118     # SimulateBus(randomness = 0.015, extratext = "randomness =
            0.015")
119     # SimulateBus(randomness = 0.025, extratext = "randomness =
            0.025")
120     # SimulateBus(randomness = 0.050, extratext = "randomness =
            0.050")
121     # SimulateBus(randomness = 0.075, extratext = "randomness =
            0.075")
122     SimulateBus(randomness = randomness2, iter = 10, extratext = "
            randomness␣=␣0.125")
123     # SimulateBus(randomness = 0.250, extratext = "randomness =
            0.250")
124     # SimulateBus(randomness = 0.500, extratext = "randomness =
            0.500")
125     # SimulateBus(randomness = 1.000, extratext = "randomness =
            1.000")
126
127 if __name__ == "__main__":
128     main()
```