信息网络中的随机量建模

引子

- **信息网络中存在很多随机量。**
  - 网络中的业务是随机的；
    - 云计算中的计算任务是随机的；
  - 移动（无线）网络中的节点位置是随机的；
  - 自组织网络中的网络拓扑结构是随机的；
  - 受各种地理条件、建筑、植被等影响，无线传播环境下的信道是随机的；
- **随机量建模对蒙特卡洛网络仿真非常重要。**
  - 输入模型对仿真模型的输出影响很大；
  - 选择适当的分布来表示输入数据是非常关键的；
  - 如何在仿真程序中实现这些随机量模型同样重要。

引子

信息网络仿真中，需要对这些随机量进行准确建模：

建模不是闭门造车，**模型从实测数据中来**；

随机量建模包含两个步骤：

- 从数据到模型：输入建模
  - 如何采集数据？
  - 如何选择正确的分布？
  - 如何估计分布参数？
  - 如何验证选择的正确性？
- 模型在仿真程序中的实现

目录

从数据到模型

- 从真实场景收集数据
- 选择用**非参数**模型还是**参数化概率**模型？

如果选择参数化概率模型

- 1、选择一个适当的**概率分布**来表示输入过程
- 2、估计概率分布**关联参数**
- 3、评估所选分布和关联参数的**拟合度**

从数据到模型

1、数据收集

2、输入建模

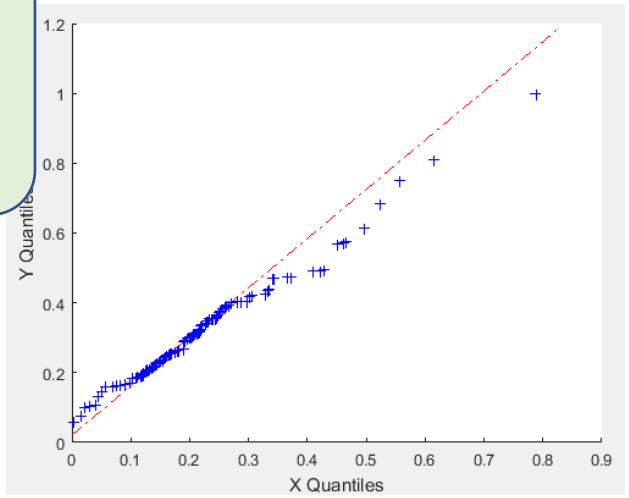
- 轨迹驱动仿真（trace-driven simulation）
  - 采集数据直接用于仿真
- 经验输入建模（empirical input modeling）
  - 从采集数据中得到随机变量用于仿真
- 理论输入建模（theoretical input modeling）
  - 理论分布函数的参数根据采集数据进行估计
  - 随机变量由拟合分布函数生成

数据收集

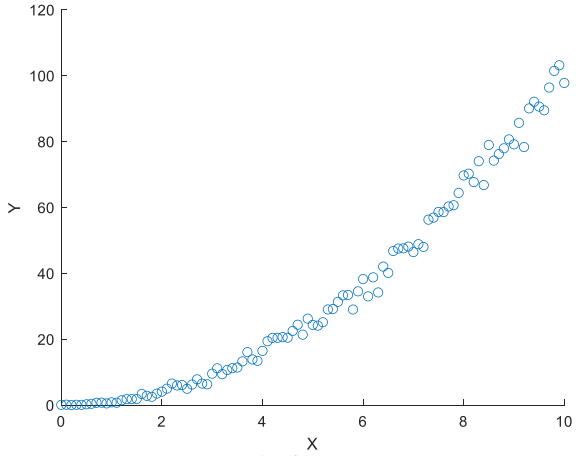
- **中位数**：把所有数值由小到大排列后，正中间的数。
- **四分位数**：把所有数值由小到大排列并分成四等份，处于三个分割点位置的数。

提高数据准确性的建议：

- 对输入过程进行**层次划分**，对不同层次的数据分别进行收集；
  - 例如HTTP业务建模为：会话级、页面级、连接级、数据包级
- 在收集数据的同时，对所收集数据的有效性进行分析，确保数据可用于仿真；
- 对不同**数据集**的**同质性**进行分析
  - 可以用Q-Q图对数据集的同质性进行分析
  - 作用：同质的数据集可以共同建模
- 通过绘制散点图的方式判断两个**变量之间**是否有**关联性**；
  - 作用：利用强关联可降低变量数



Q-Q图：横纵坐标分别为两个数据集的分位数



散点图

从数据到模型

1、数据收集

2、输入建模

- 轨迹驱动仿真（trace-driven simulation）
  - 采集数据直接用于仿真
- 经验输入建模（empirical input modeling）
  - 从采集数据中得到随机变量用于仿真
- 理论输入建模（theoretical input modeling）
  - 理论分布函数的参数根据采集数据进行估计
  - 随机变量由拟合分布函数生成

经验建模

1、非参数建模(Nonparametric Modeling)

2、个体数据的经验建模(Empirical Modeling of Individual Data)

3、分组数据的经验建模(Empirical Modeling of Grouped Data)

非参数建模

非参数建模：生成的随机输入为从收集数据中以$1/n$的概率进行采样得到的数据，其中n为收集数据的总数。

建模方式：（收集数据为$\{X_k: k = 1 \sim n\}$）

1、生成一个随机均匀分布数$u \sim U(0,1)$;

2、令$P = n \times u$，则$k = [P] + 1$;

3、X_k即为生成的随机输入。

非参数建模

例：假设已收集到某银行ATM机在一个小时内前十名顾客的服务时间数据，如下表所示。如何用非参数建模方式进行建模？

观察数	1	2	3	4	5	6	7	8	9	10
服务时间	56	51	73	65	84	58	62	69	44	66

建模方式：

1、生成一个随机均匀分布数如，$u = 0.369981;$

2、$P = n \times u = 3.69981$，则$k = [P] + 1 = 4;$

3、$X_4 = 65$ 即为生成的随机输入。

经验建模

1、非参数建模(Nonparametric Modeling)

2、个体数据的经验建模(Empirical Modeling of Individual Data)

3、分组数据的经验建模(Empirical Modeling of Grouped Data)

个体数据的经验建模

个体数据的经验建模：将收集到的数据进行递增排列，经验函数为分段线性函数。$F(X_k)=(k-1)/(n-1)$。生成的随机输入x为随机生成的0-1之间的数据在分段函数上对应的值，如图所示。

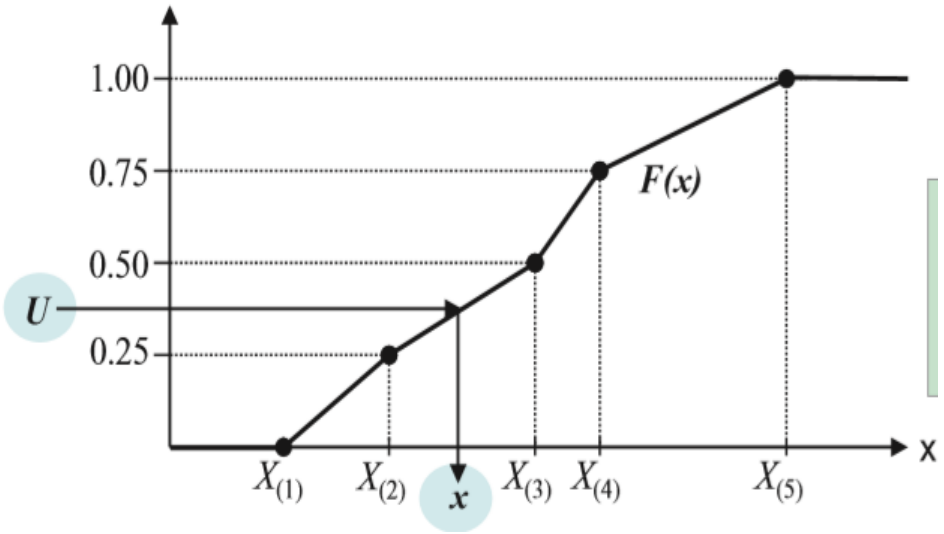
建模方式：（收集数据为$\{X_k: k=1\sim n\}$）

1、对数据以递增方式排列，
　　然后绘制分段函数$F(X)$;

2、生成一个随机均匀分布数$u\sim U(0,1)$;

3、令$P=(n-1)\times u$，则$J=[P]+1$;

4、$x=X_{(J)}+(P-J+1)\times\left(X_{(J+1)}-X_{(J)}\right)$。



个体数据的经验建模

例：假设已收集到某银行ATM机在一个小时内前十名顾客的服务时间数据，如下表所示。如何用个体数据的输入建模方式进行输入建模？

观察数	1	2	3	4	5	6	7	8	9	10
服务时间	56	51	73	65	84	58	62	69	44	66

建模方式：

1、对数据大小进行重新排列,并进行分段函数的绘制；

观察数	1	2	3	4	5	6	7	8	9	10
服务时间	44	51	56	58	62	65	66	69	73	84

2、生成一个随机均匀分布数$u = 0.369981$;

3、 $P = (n-1)\times u = 3.328929$ ，则$\mathrm{J} = [P] + 1 = 4$;

4、 $x = X_{(J)} + (P - J + 1)\times\left(X_{(J+1)} - X_{(J)}\right) = 59.316$即为生成的随机输入。

经验建模

1、非参数建模(Nonparametric Modeling)

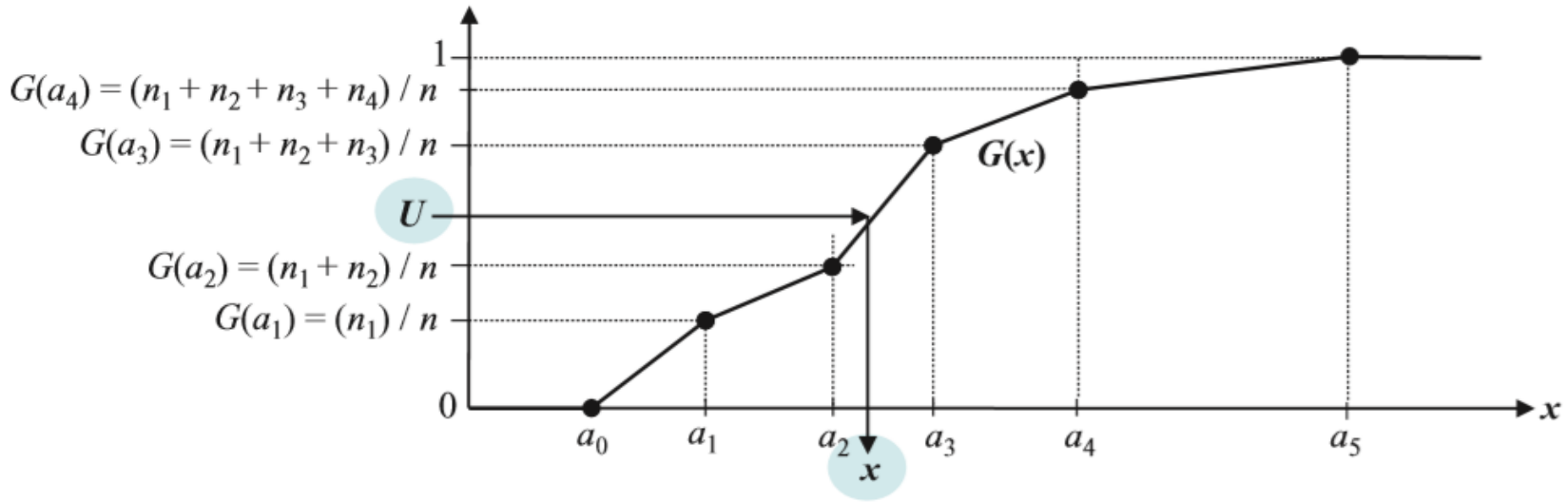
2、个体数据的经验建模(Empirical Modeling of Individual Data)

3、分组数据的经验建模(Empirical Modeling of Grouped Data)

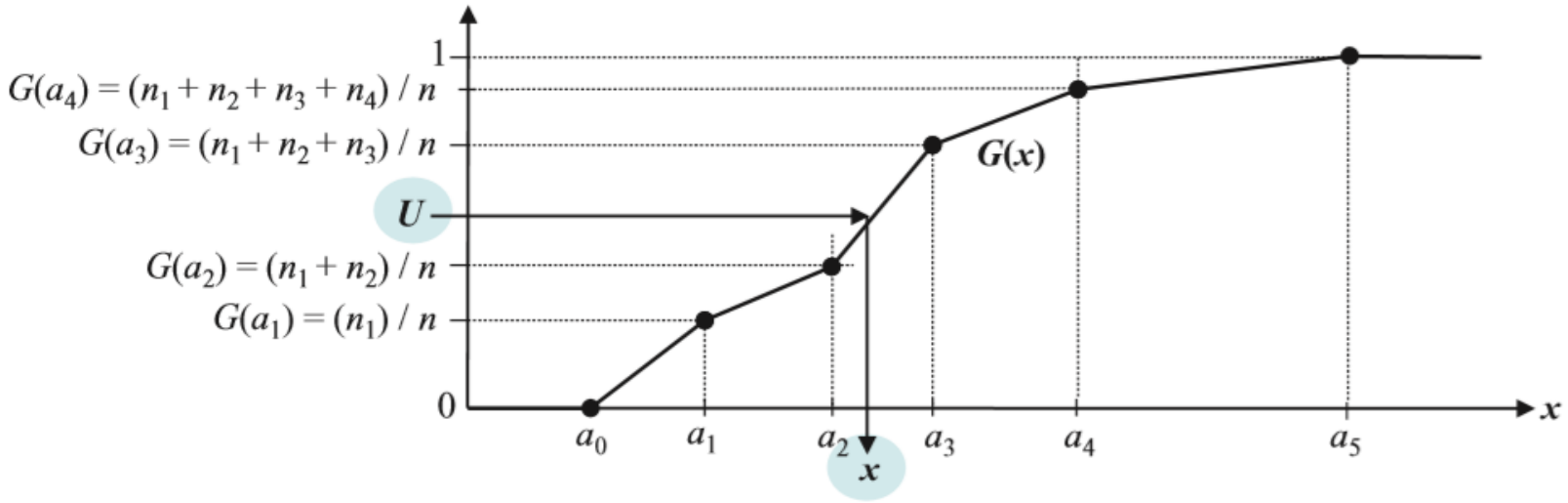
分组数据的经验建模

分组数据的经验建模：

- 将收集的数据分为m个区间$\{[a_0,a_1),[a_1,a_2),\dots,[a_{m-1},a_m)\}$，第$j$个区间包含$n_j$个数据
- 定义经验函数为分段函数：$G(a_0)=0$，$G(a_j)=\sum_{i=1}^{j} n_i/n$，其中$j=1\sim m$，$n=\sum n_j$。
- 生成随机输入x：随机生成的0-1之间的数据在分段函数上对应的值，如图所示（图为m=5的情况）。



分组数据的经验建模



建模方式：（收集数据为 $\{X_k: k=1\sim n\}$）

1、以一个合适的区间大小对收集到的数据进行分组$\{[a_0,a_1),[a_1,a_2),\ldots,[a_{m-1},a_m)\}$，并计算每个组所含有的数据数$n_j$，根据数据个数占比绘制经验函数$G(X)$;

2、生成一个随机均匀分布数$u\sim U(0,1)$;

3、使J满足$G(\mathrm{a}_J)\le u\le G(\mathrm{a}_{J+1})$;

4、$x=\mathrm{a}_J+[u-G(\mathrm{a}_J)]\times(\mathrm{a}_{J+1}-\mathrm{a}_J)/[G(\mathrm{a}_{J+1})-G(\mathrm{a}_J)]$。

分组数据的经验建模

例：假设已收集到某银行ATM机在一个小时内前十名顾客的服务时间数据，如下表所示。如何用分组数据的输入建模方式进行输入建模？

观察数	1	2	3	4	5	6	7	8	9	10
服务时间	56	51	73	65	84	58	62	69	44	66

建模方式：

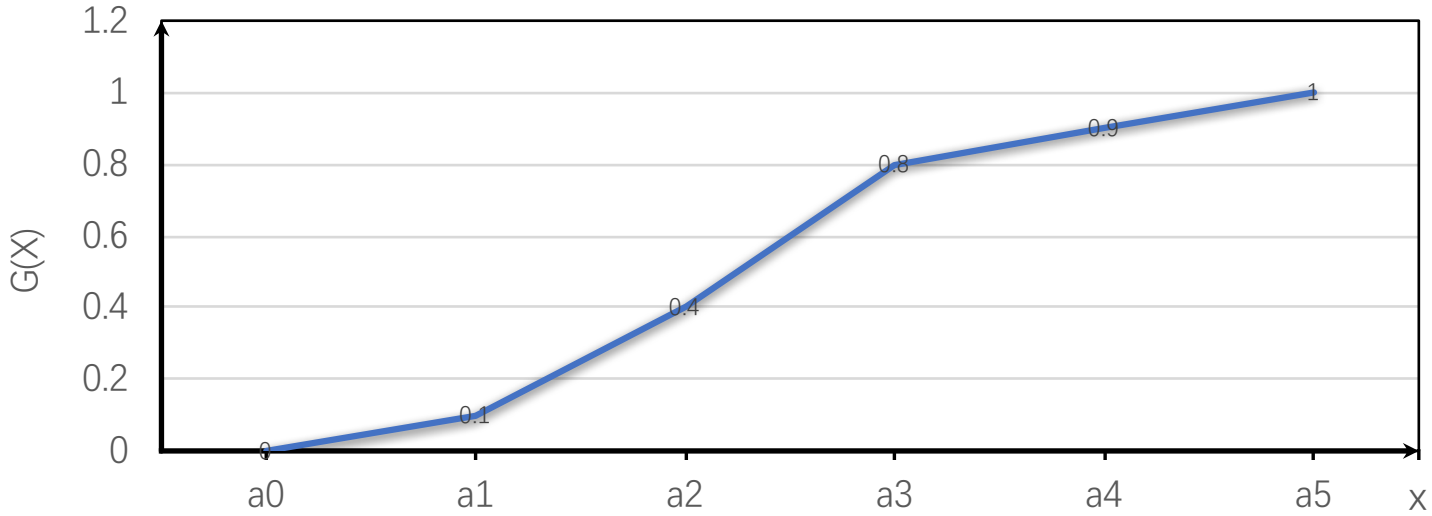
随机数以40%概率位于60-70区间内

1、对数据大小进行分组，区间大小为10，分为5组并进行分段函数的绘制；

组号	1	2	3	4	5
区间	40-50	50-60	60-70	70-80	80-90
数据数	1	3	4	1	1
G(X)	1/10	4/10	8/10	9/10	10/10

$$a_0 = 40; a_1 = 50; a_2 = 60; a_3 = 70; a_4 = 80; a_5 = 90$$

分组数据的经验建模



2、生成一个随机均匀分布数$u = 0.369981$;

3、使J满足$G(\mathrm{a}_J) \le u \le G(\mathrm{a}_{J+1})$，$J = 1$;

4、$x = \mathrm{a}_J + [u - G(\mathrm{a}_J)] \times \frac{\mathrm{a}_{J+1} - \mathrm{a}_J}{[G(\mathrm{a}_{J+1}) - G(\mathrm{a}_J)]} = 58.9960$。

理论建模

1、数据独立性检查

2、分布函数的选择

3、参数估计

4、拟合度测试

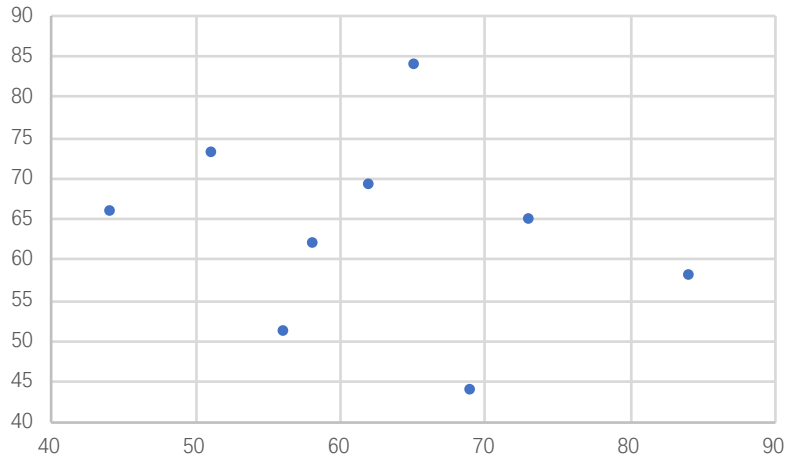
数据独立性检查

简单直观的**独立性**检查方式：

将收集的**一组数据**按照时间收集顺序进行排序，$X_1, X_2, \ldots, X_n$，以X_i为横坐标，X_{i+1}为纵坐标绘制散点图。若散点图为随机分布，则数据独立。

例：假设已收集到某银行ATM机在一个小时内前十名顾客的服务时间数据，如下表所示。判断数据独立性。

观察数	1	2	3	4	5	6	7	8	9	10
服务时间	56	51	73	65	84	58	62	69	44	66



如左图所示，图中数据为随机分布，所以收集到的数据具有独立性。

（这里只考虑了前后两个数据之间的独立性，如果需要，可以考虑X_i和X_{i+k}之间的相关性）

数据独立性检查

- 数据不独立情况下的处理方法：
  - 如果检测到数据不独立，首先应当对所对应的物理过程进行分析，**判断**数据间本身是否就应当具有相关性；
  - 如果是，如下两种建模方法：

如果**数据间本身就应具有相关性**，则应按上一章的随机过程建模方法进行建模；

如果**物理过程可分割为多个不同层次或子过程**，则可以将整个过程划分为层次或子过程，使得划分后的数据统计具有独立性。

理论输入建模

1、数据独立性检查

2、分布函数的选择

3、参数估计

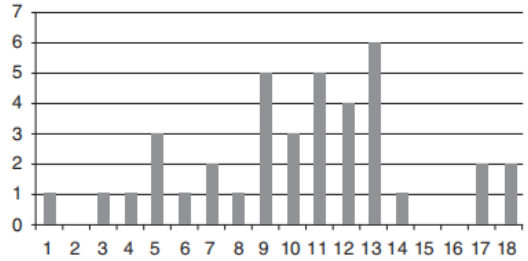
4、拟合度测试

分布函数选择

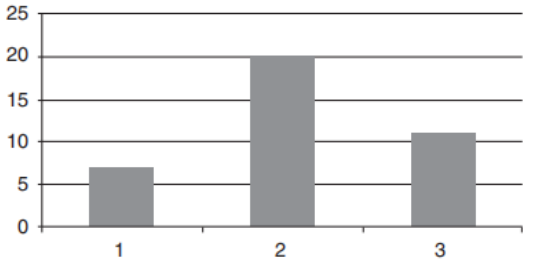
- 分布函数选择：基于理论判断和直方图的形状。
- 在绘制直方图时注意选择合适的直方图区间大小。

  一般选择直方图区间数 $=\sqrt{(有效数据数)}$

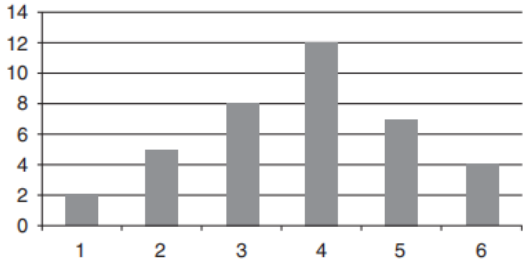
  区间过大或过小都无法反映正确的形状



区间过小



区间过大



区间大小适中

分布函数选择

- 选择分布函数时，需考虑数据：
  - 连续 or 离散？

    常见**连续**分布：均匀分布、正态分布、指数分布、t-分布、Gamma分布等

    常见**离散**分布：伯努利分布、泊松分布、二项分布、负二项分布等

  - 是否非负？
  - 是否关于均值对称？

分布函数选择

- 一些特殊的度量值
  - Lexis 比率（Lexis ration）τ：方差与均值之比，用于区分离散分布的类型
    - 数据越**分散**， τ 越大
    - 例如：泊松分布τ =1， 二项分布τ<1， 负二项分布τ>1
  - 变异系数（Coefficient of variance)：标准差与均值之比， 常用于连续分布
    - 例如：指数分布该值等于1
  - 偏斜：对称性的度量$s = \frac{E(X-EX)^3}{\sigma^3}$
    - 偏斜**接近0**：正态分布、均匀分布等对称分布
    - 偏斜**较大**：指数分布、weibull分布、帕累托分布、瑞利分布等

根据上述数据特征，选择合适的分布函数

理论输入建模

1、数据独立性检查

2、分布函数的选择

3、参数估计

4、拟合度测试

参数估计

最大似然估计：指数分布，正态分布，对数正态分布；

矩量法：Erlang分布，Beta分布。

输入变量类型	分布	参数估计
到达时间间隔	指数分布(θ)	最大似然估计
	Erlang分布(k,θ)	矩量法
服务(修复)时间	Beta分布(α,β)	矩量法
	正态分布(μ,σ)	最大似然估计
	对数正态分布(μ,σ)	最大似然估计

参数估计

➢ 最大似然估计法：

- 基本思想：利用已知的样本结果信息$x_1, x_2, \ldots, x_n$，反推最具有可能（最大概率）导致这些样本结果出现的模型参数值θ 。
- 似然函数：

$$lik(\theta) = f_D(x_1, x_2, \ldots, x_n|\theta)$$

其中，f_D为概率密度函数

- 最大似然求θ ：使**似然函数最大**的参数值θ

参数估计

最大似然举例：正态分布，密度函数为 $N\left(x|\mu,\sigma^2\right)=\frac{1}{\left(2\pi\sigma^2\right)^{\frac{1}{2}}}\exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$

数据 $\boldsymbol{x}=(x_1,x_{2,}\ldots,x_N)^T$的联合概率为 $p\left(\boldsymbol{x}|\mu,\sigma^2\right)=\prod_{i=1}^{N}N\left(x_i|\mu,\sigma^2\right)$

- 步骤一：将正态分布**代入对数似然函数**

$$\ln L=\ln p\left(\boldsymbol{x}|\mu,\sigma^2\right)=-\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2-\frac{N}{2}\ln\sigma^2-\frac{N}{2}\ln(2\pi)$$

- 步骤二：对似然函数求**偏导**

$$\begin{cases}\frac{\partial\ln L}{\partial\mu}=\frac{1}{\sigma^2}\sum_{i=1}^{N}\left(x_i-\mu\right)=0\\ \frac{\partial\ln L}{\partial(\sigma^2)}=-\frac{N}{2\sigma^2}+\frac{1}{2\sigma^4}\sum_{i=1}^{N}\left(x_i-\mu\right)^2=0\end{cases}$$

参数估计

- 矩量法：

基本思想：计算所采集数据各阶中心矩的统计值**t**，以及所选分布的矩f(**s**)，其中**s**为参数集合。设置$f(\mathbf{s})=\mathbf{t}$，通过解方程得到相应参数**s**。

矩量法举例：爱尔兰分布，密度函数为$f(x)=\frac{\theta^{-k}x^{k-1}e^{-x/\theta}}{(k-1)!}$。

- 步骤一：计算**所采集数据各阶中心矩**的统计值。

一阶样本矩：$m_1=\frac{1}{n}\sum_{i=1}^{n}x_i$；二阶样本矩：$m_2=\frac{1}{n}\sum_{i=1}^{n}x_i^2$

- 步骤二：**根据密度函数**得到一阶和二阶原点矩。

$$E[X]=\int xf(x)dx=k\theta;\qquad E[X^2]=\int x^2f(x)dx=k(k+1)\theta^2$$

- 步骤三：**建立等式**求解

$$\hat{k}=\frac{(m_1)^2}{[m_2-(m_1)^2]};\qquad \hat{\theta}=\frac{[m_2-(m_1)^2]}{m_1};$$

理论输入建模

1、数据独立性检查

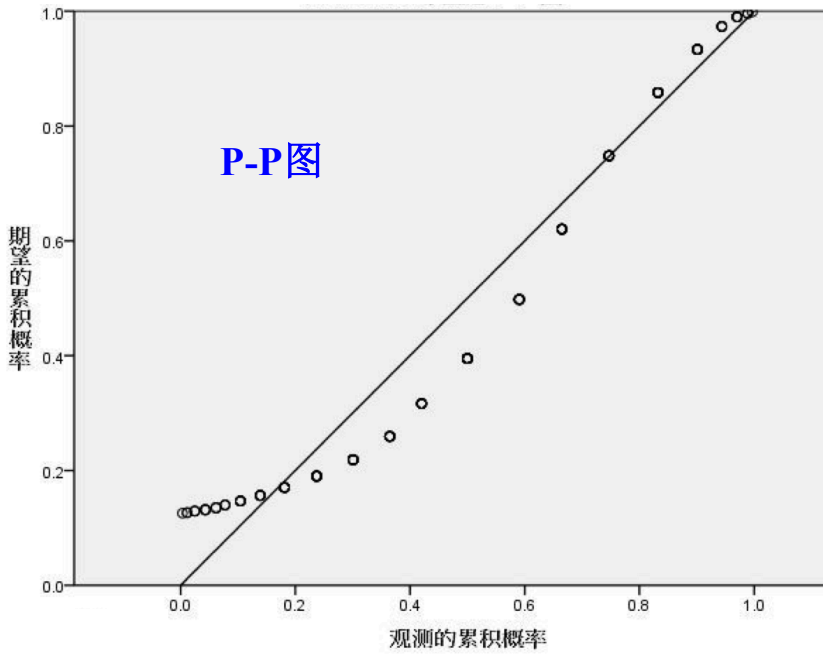
2、分布函数的选择

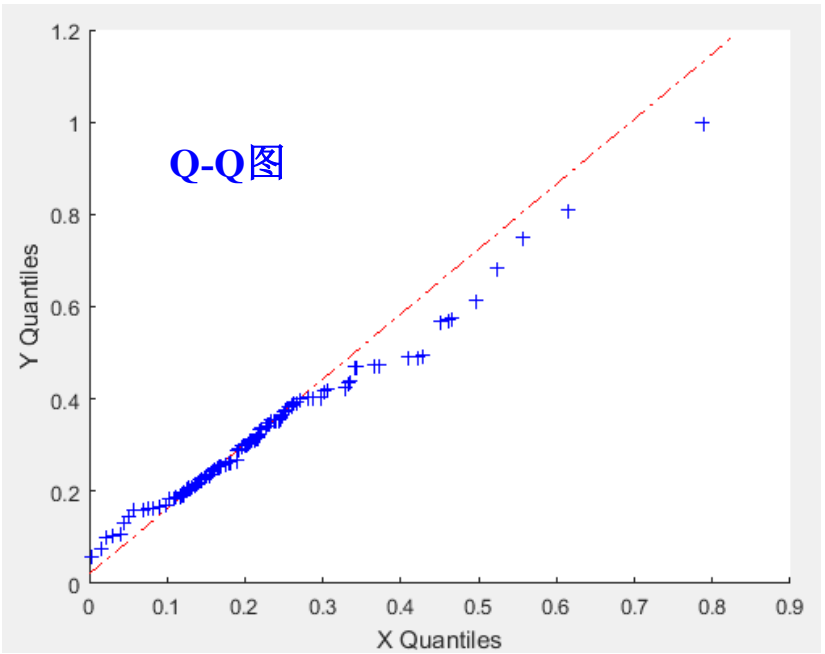
3、参数估计

4、拟合度测试

拟合度测试

- 拟合度测试方法有图形法和解析法两种
- 图形法：
  - P-P图：数据CDF和拟合分布CDF的对应点分别作为横纵坐标
  - Q-Q图：数据分位点和拟合分布分位点分别作为横纵坐标





拟合度测试

- 解析法：
  - 柯尔莫可洛夫-斯米洛夫检验（Kolmogorov–Smirnov test，K-S test）：检验单一样本是否**来自某一特定分布**的方法

    - 定义：$F_0(x)$特定分布的分布函数，$F_n(x)$表示一组随机样本计算得到的累积概率函数。
    - K-S值：

      $$D = \max| F_0(x) - F_n(x) |$$

      当实际观测D小于门限，认为服从特定分布

  - 卡方检验：比较**理论频数**和**实际频数**的吻合程度

$$X^2 = \sum \frac{(\text{observed} - \text{expected})^2}{\text{expected}}$$

拟合度测试

拟合度测试方法：卡方检验

- 卡方检验方式：将采集数据分为m组$\{[a_0, a_1), [a_1, a_2), \ldots, [a_{m-1}, a_m)\}$，第$j$个区间包含$n_j$个数据，检验统计量：

$$\chi^2 = \sum_{j=1}^{m} \frac{(n_j - np_j)^2}{n_j}, \quad p_j = \int_{a_{j-1}}^{a_j} \hat{f}(x)dx$$

其中n为有效数据量总数，$\hat{f}(x)$为拟合分布的密度函数。

- 在计算完检验统计量χ^2后，与（m-1）自由度的卡方值进行检验。
- 卡方检验条件：***n*足够大，*np_j*不能太小**。一般要求$\boldsymbol{n \geq 50}$以及$\boldsymbol{np_j \geq 5}$。当np_j不足够大时，可以适当将某些区间合并来满足这个条件。

目录

业务源建模简介

- **业务源模型重要性**：在信息网络建模中，业务源的模型至关重要，准确的业务模型能够对业务中的关键特征进行表示。
- 不同的实际应用将产生不同的业务源类型，例如：**http业务、P2P业务、话音业务、视频业务和计算业务**等。
- 建模过程通常包括两步：
  - 首先：通过适当的方法来测量关键特征，方法如：服务器日志、客户端日志、数据包跟踪等方法；
  - 然后：在模型中表示出这些特征，例如采用马尔可夫链等方法。

业务源模型的共性特征

- **层次性**：基本都将业务过程分为多个不同层次，分层建模
- **多参数：**
  - 基本都不是单参数模型
  - 通过对时间间隔、持续时间、数据量大小等多个量分别建模，最终达到模拟整个随机过程的目的
- **协议流程分析：**
  - 业务源模型往往与应用层协议密切相关，这些协议的处理流程基本都遵循一定标准，通过对标准所规定的协议流程的分析，从中划分出**确知量**和真正的**随机量**
  - 直接对这些随机量进行建模，对确知量则直接通过模拟流程得到。

HTTP业务

1.1.1、HTTP业务模型

1.1.2、HTTP参数

1.1.3、NS3中实现HTTP业务

HTTP业务

- WWW业务：因特网的重要部分
- WWW业务**基本概念**：
  - 网页：网络业务建模的重要模块
    - 包括ASCII形式的超文本标记语言（HTML）
    - HTML：定义网页的结构和互联关系
  - 主对象：组成网页页面的主体框架，即一个由HTML描述的文档
  - 嵌入对象：内嵌在页面框架中的文本、声音或者图片等。

```
<object width="400" height="400" data="helloworld.swf"></object>
：嵌入 一段 .swf
```

  - 下载网页的过程：向网页的URL链接发送请求，通过**HTTP协议**，与网页服务器之间建立TCP连接

HTTP业务

HTTP业务模型

首先需要收集重要的数据来进行分析，以确定适当的参数

然后进行建模

数据采集方法

- **服务器日志：**
  - 服务器跟踪所服务的文件
  - 没有考虑用户行为
- **客户端日志：**
  - 在客户端进行文件跟踪，需要修改浏览器，目前一般不可用
- **数据包追踪：**
  - 最流行的方法
  - 如：Wireshark、TCP-Dump等工具

建模方法

- **面向页面的模型**

面向页面的建模

面向页面的模型通常以**多层结构**来进行描述，包括会话级、页面级、连接级、数据包级。

- **会话级**：
  - 描述用户的行为
  - 创建一个页面后执行的跳转、重定向、整合等都可以是一个会话，只要不关闭浏览器页面，就是一次会话
  - 在会话中用户浏览多个网页，会话表示为页面的集合
  - 参数：
    - 例如：每个周期的网络会话个数、该周期中的会话分布
- **页面级**：
  - 描述每个会话的网页
  - 参数
    - 例如：每个会话的网页数，两个网页之间时间的分布
    - 每个页面之间的时间建模为阅读时间

HTTP业务

面向页面的建模

面向页面的模型通常以**多层结构**来进行描述，包括会话级、页面级、连接级、数据包级。

- **连接级**：
  - 网页包括多个目标，通过多个TCP连接进行传输
  - 参数例如：描述每个页面的连接个数，两个连续连接之间的时间，连接大小的分布
- **数据包级**：
  - 在TCP/IP数据包中对每个连接的字节进行划分
  - 参数例如：描述数据包大小的分布，数据包到达的时间间隔。

HTTP业务

1.1.1、HTTP业务模型

1.1.2、HTTP参数

1.1.3、NS3中实现HTTP业务

HTTP参数：会话级

会话级重要参数包括：

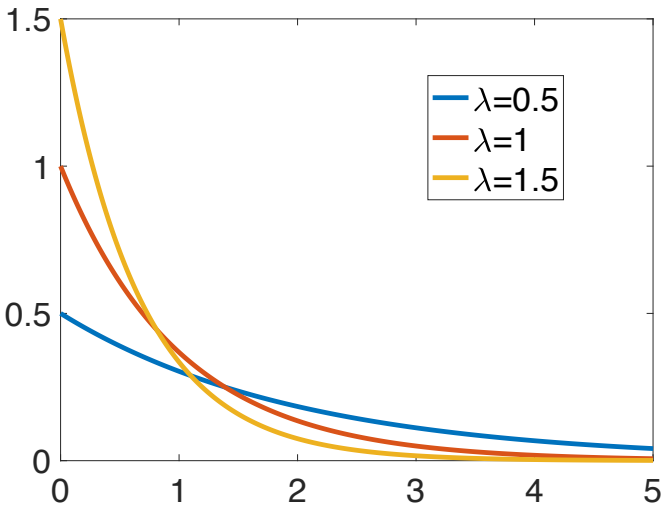
- 会话到达间隔
- 浏览时间
- 每个会话的页面数

- 会话到达间隔的常用分布：
  - 指数分布
  - 帕累托分布
- 浏览时间的常用分布：
  - Weibull分布
  - Gamma分布
  - 几何分布
- 每个会话页面数的常用分布：
  - Weibull分布
  - Lognormal分布
  - 几何分布等

HTTP参数：会话级

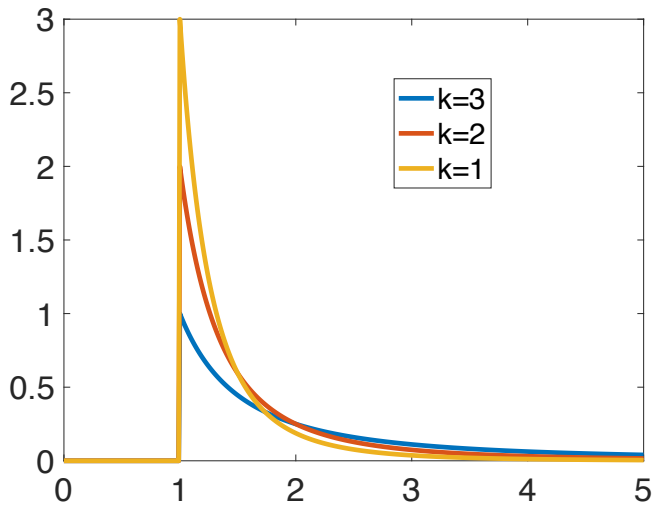
指数分布

$$f(x)=\begin{cases}\lambda e^{-\lambda x} & x>0\\ 0 & x\le 0\end{cases}$$



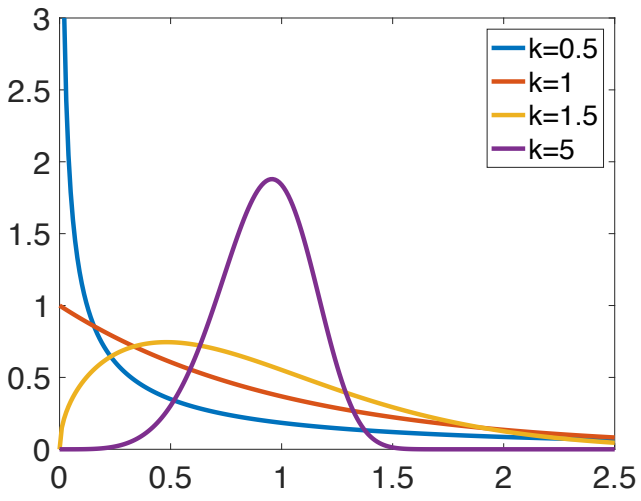
帕累托分布

$$p(x)=\begin{cases}0, & if\ x<x_{min}+\mu;\\ \dfrac{kx_{min}^{k}}{(x-\mu)^{k+1}}, & if\ x\ge x_{min}+\mu\end{cases}$$



Weibull分布

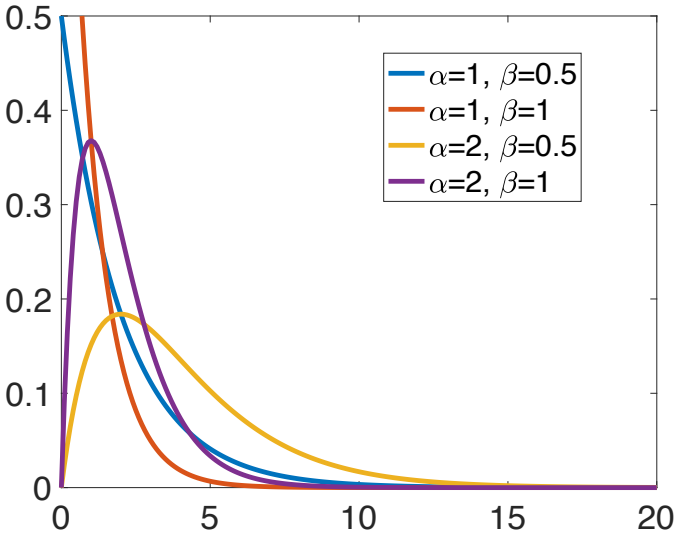
$$f(x;\lambda,k)=\begin{cases}\dfrac{k}{\lambda}\left(\dfrac{x}{\lambda}\right)^{k-1}e^{-(x/\lambda)^{k}} & x\ge 0\\ 0 & x<0\end{cases}$$



HTTP参数：会话级

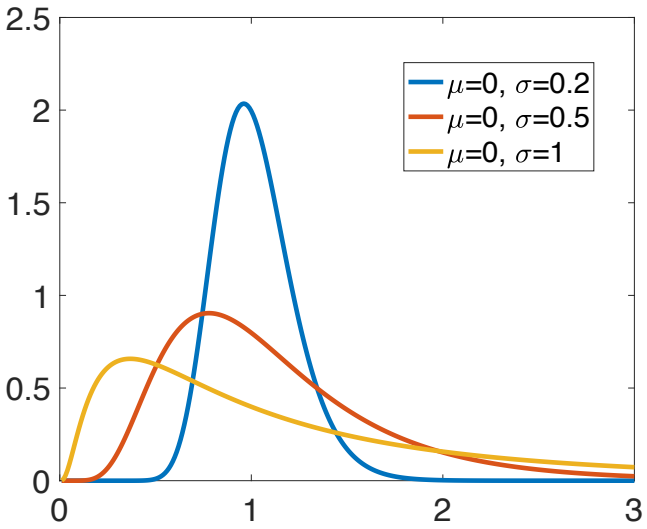
Gamma分布

$$f(x,\beta,\alpha)=\frac{\beta^{\alpha}}{\Gamma(\alpha)}x^{\alpha-1}e^{-\beta x},x>0$$



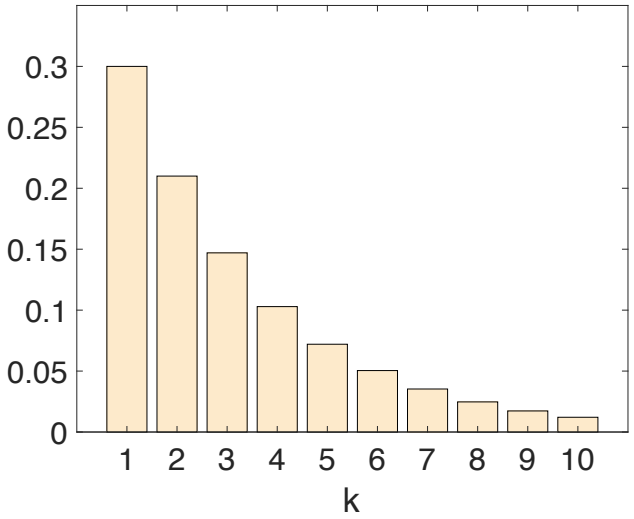
对数正态分布

$$p(x)=\frac{1}{x\sigma\sqrt{2\pi}}\mathrm{e}^{-\frac{(\ln x-u)^2}{2\sigma^2}}$$



几何分布

$$P(\mathrm{X}=k)=(1-p)^{\mathrm{k}-1}\mathrm{p},k=1,2,\ldots$$



HTTP参数：页面级

页面级重要参数包括：

- 同一会话中两个连续页面间时间
- 主对象特征
- 嵌入对象特征
- 主对象解析时间

- 同一会话中两个连续页面间时间的常用分布：
  - Weibull分布
  - Gamma分布
  - 帕累托分布
  - 几何分布
- 主对象大小的常用分布：
  - Lognormal分布
  - Pareto分布
- 嵌入对象个数的常用分布：
  - Gamma分布
  - Pareto分布
- 主对象解析时间的常用分布：
  - Weibull分布
  - 指数分布

HTTP参数：连接级

连接级重要参数包括：

- 每个页面的连接个数
- 同一页面两个连续连接的时间间隔
- 连接的大小

- 连接数的常用分布：
  - Gamma分布
  - 帕累托分布
- 时间间隔的常用分布：
  - Gamma分布
- 连接大小的常用分布：
  - Lognormal分布

HTTP业务

1.1.1、HTTP业务模型

1.1.2、HTTP参数

1.1.3、NS3中实现HTTP业务

HTTP业务

01 业务产生器

包括了一到多个ThreeGppHttpClient应用，连接到一个ThreeGppHttpServer应用

02 客户端模型

对网页浏览器进行建模，从服务器请求网页

03 服务器模型

- 负责服务所请求的网页。
- 服务器基于所收到的请求类型，发送主对象（即网页的HTML文件）或者嵌入对象（例如HTML文件的图像）

3GPP HTTP客户端

- 该客户端是一种对网页服务器业务进行仿真的应用，该应用与**ThreeGppHttpServer**应用一起工作。
- 工作流程：

1. 打开到目标网络服务器的连接
2. 连接建立后，立即发送请求数据包，从服务器请求主对象
3. 在收到主对象后，对其进行解析。
4. 解析过程确定网页中嵌入式对象的个数。
   - 如果有至少一个嵌入式对象，则从服务器请求第一个嵌入式对象。而后续嵌入式对象的请求会跟随前一个完全接收到的对象。
   - 如果没有嵌入式对象，则该应用进入阅读时间。
5. 阅读时间需要一个较长的随机时延，此时不进行任何网络业务。
6. 阅读时间结束后，返回第2步。

3GPP HTTP服务器

- 对网页服务器业务进行仿真的应用，与**ThreeGppHttpClient**一起工作
- 响应请求：
  - 每个请求是一个数据包，包括了**ThreeGppHttpHeader**
  - 其中的属性content type决定了客户端所请求对象的类型，可能是主对象类型，或嵌入式对象类型
- 产生返回客户端的对象：
  - 所产生的对象大小是随机产生的
  - 每个对象可以以多个数据包的形式进行发送。
- **ThreeGppHttpServerTxBuffer**
  - 每个实体跟踪所服务的对象类型，以及剩下需要发送的字节数。
  - 该应用接收来自客户端的连接请求，每个连接一直处于保持状态直到客户端断开连接
  - **ThreeGppHttpVariables**配置最大传输单元大小。默认为536字节，最大为1460字节

3GPP HTTP——代码实现

3GPP HTTP客户端——分析主对象时间

分析主对象时间服从指数分布：

```
const Time parsingTime = m_httpVariables->GetParsingTime ();
```

```
ThreeGppHttpVariables::GetParsingTime ()
{
  return Seconds (m_parsingTimeRng->GetValue ());
}
```

```
m_parsingTimeRng = CreateObject<ExponentialRandomVariable> ();
```

指数分布的均值设置：

```
void
ThreeGppHttpVariables::SetParsingTimeMean (Time mean)
{
  NS_LOG_FUNCTION (this << mean.GetSeconds ());
  m_parsingTimeRng->SetAttribute ("Mean", DoubleValue (mean.GetSeconds ()));
}
```

3GPP HTTP客户端——主对象中嵌入对象数量

主对象中嵌入对象数量服从Pareto分布：

```
m_embeddedObjectsToBeRequested = m_httpVariables->GetNumOfEmbeddedObjects ();
ThreeGppHttpVariables::GetNumOfEmbeddedObjects ()
{// 参数验证
  const uint32_t upperBound =
    static_cast<uint32_t> (m_numOfEmbeddedObjectsRng->GetBound ());
  if (upperBound <= m_numOfEmbeddedObjectsScale){
      NS_FATAL_ERROR ("`NumOfEmbeddedObjectsMax` attribute "
                      << " must be greater than"
                      << " the `NumOfEmbeddedObjectsScale` attribute.");}
  //找一个随机值使它落入[scale, upperBound)区间，上面验证保证不会无限循环。
  uint32_t value;
  do{
      value = m_numOfEmbeddedObjectsRng->GetInteger ();
    }
  while ((value < m_numOfEmbeddedObjectsScale) || (value >= upperBound));
//返回值在区间[0, (upperBound - scale))内。
  return (value - m_numOfEmbeddedObjectsScale);}

m_numOfEmbeddedObjectsRng = CreateObject<ParetoRandomVariable> ();
```

3GPP HTTP客户端——主对象中嵌入对象数量

Pareto分布的max(μ)、shape($1/k$)、scale(x_{min})设置：

$$p(x) = \begin{cases} 0, & if\ x < x_{min} + \mu; \\ \dfrac{k x_{min}^{k}}{(x-\mu)^{k+1}}, & if\ x \geq x_{min} + \mu \end{cases}$$

```
void
ThreeGppHttpVariables::SetNumOfEmbeddedObjectsMax (uint32_t max)
{ NS_LOG_FUNCTION (this << max);
  m_numOfEmbeddedObjectsRng->SetAttribute ("Bound",
                                           DoubleValue (static_cast<double> (max)));}
void
ThreeGppHttpVariables::SetNumOfEmbeddedObjectsShape (double shape)
{ NS_LOG_FUNCTION (this << shape);
  NS_ASSERT_MSG (std::fabs (shape - 1.0) > 0.000001,
                 "Shape parameter must not equal to 1.0.");
  m_numOfEmbeddedObjectsRng->SetAttribute ("Shape", DoubleValue (shape));}
void
ThreeGppHttpVariables::SetNumOfEmbeddedObjectsScale (uint32_t scale)
{NS_LOG_FUNCTION (this << scale);
  NS_ASSERT_MSG (scale > 0, "Scale parameter must be greater than zero.");
  m_numOfEmbeddedObjectsScale = scale;
  m_numOfEmbeddedObjectsRng->SetAttribute ("Scale", DoubleValue (scale));}
```

3GPP HTTP客户端——阅读时间

阅读时间服从指数分布：

```
const Time readingTime = m_httpVariables->GetReadingTime ();
ThreeGppHttpVariables::GetReadingTime ()
{
  return Seconds (m_readingTimeRng->GetValue ());
}
m_readingTimeRng = CreateObject<ExponentialRandomVariable> ();
```

指数分布的均值设置：

```
void
ThreeGppHttpVariables::SetReadingTimeMean (Time mean)
{
  NS_LOG_FUNCTION (this << mean.GetSeconds ());
  m_readingTimeRng->SetAttribute ("Mean", DoubleValue (mean.GetSeconds ()));
}
```

3GPP HTTP服务端——MTU大小

MTU大小服从取值为1500和576的离散分布：

```
m_mtuSize = m_httpVariables->GetMtuSize ();
```

```
ThreeGppHttpVariables::GetMtuSize ()
{const double r = m_mtuSizeRng->GetValue ();
  NS_ASSERT (r >= 0.0);
  NS_ASSERT (r < 1.0);
  if (r < m_highMtuProbability) {
      return m_highMtu; // 1500 bytes if including TCP header.}
  else {
      return m_lowMtu; // 576 bytes if including TCP header. }}
```

```
m_mtuSizeRng = CreateObject<UniformRandomVariable> ();
```

3GPP HTTP服务端——MTU大小

MTU大小涉及到的代码参数（最大MTU、最小MTU、选择大MTU的概率）设置：

```
.AddAttribute ("LowMtuSize",
               "The lower MTU size.",
               UintegerValue (536),
               MakeUintegerAccessor (&ThreeGppHttpVariables::m_lowMtu),
               MakeUintegerChecker<uint32_t> (0))
.AddAttribute ("HighMtuSize",
               "The higher MTU size.",
               UintegerValue (1460),
               MakeUintegerAccessor (&ThreeGppHttpVariables::m_highMtu),
               MakeUintegerChecker<uint32_t> (0))
.AddAttribute ("HighMtuProbability",
               "The probability that higher MTU size is used.",
               DoubleValue (0.76),
               MakeDoubleAccessor (&ThreeGppHttpVariables::m_highMtuProbability),
               MakeDoubleChecker<double> (0, 1))
```

3GPP HTTP服务端——主对象大小

主对象大小服从对数正态分布：

```
const uint32_t objectSize = m_httpVariables->GetMainObjectSize ();
```

```
ThreeGppHttpVariables::GetMainObjectSize ()
{// 参数验证.
  if (m_mainObjectSizeMax <= m_mainObjectSizeMin)
    {NS_FATAL_ERROR ("`MainObjectSizeMax` attribute "
                     << " must be greater than"
                     << " the `MainObjectSizeMin` attribute.");  }
  //寻找一个值使它落在[min, max)区间内
  uint32_t value;
  do
    { value = m_mainObjectSizeRng->GetInteger (); }
  while ((value < m_mainObjectSizeMin) || (value >= m_mainObjectSizeMax));
  return value;}
```

```
m_mainObjectSizeRng = CreateObject<LogNormalRandomVariable> ();
```

3GPP HTTP服务端——主对象大小

对数正态分布参数（均值、方差）与代码中涉及的最大最小值的设置：

```
.AddAttribute ("MainObjectSizeMean",
               "The mean of main object sizes (in bytes).",
               UintegerValue (10710),
               MakeUintegerAccessor (&ThreeGppHttpVariables::SetMainObjectSizeMean),
               MakeUintegerChecker<uint32_t> ())
.AddAttribute ("MainObjectSizeStdDev",
               "The standard deviation of main object sizes (in bytes).",
               UintegerValue (25032),
               MakeUintegerAccessor (&ThreeGppHttpVariables::SetMainObjectSizeStdDev),
               MakeUintegerChecker<uint32_t> ())
.AddAttribute ("MainObjectSizeMin",
               "The minimum value of main object sizes (in bytes).",
               UintegerValue (100),
               MakeUintegerAccessor (&ThreeGppHttpVariables::m_mainObjectSizeMin),
               MakeUintegerChecker<uint32_t> (22))
.AddAttribute ("MainObjectSizeMax",
               "The maximum value of main object sizes (in bytes).",
               UintegerValue (2000000), // 2 MB
               MakeUintegerAccessor (&ThreeGppHttpVariables::m_mainObjectSizeMax),
               MakeUintegerChecker<uint32_t> ())
```

3GPP HTTP服务端——主对象大小

对数正态分布参数（均值、方差）设置：

```
void
ThreeGppHttpVariables::SetMainObjectSizeMean (uint32_t mean)
{ NS_LOG_FUNCTION (this << mean);
  NS_ASSERT_MSG (mean > 0, "Mean must be greater than zero.");
  m_mainObjectSizeMean = mean;
  if (IsInitialized ())
    { UpdateMainObjectMuAndSigma ();}}
void
ThreeGppHttpVariables::SetMainObjectSizeStdDev (uint32_t stdDev)
{ NS_LOG_FUNCTION (this << stdDev);
  m_mainObjectSizeStdDev = stdDev;
  if (IsInitialized ())
    {UpdateMainObjectMuAndSigma ();}}
```

3GPP HTTP服务端——主对象大小

对数正态分布参数更新：

```
void
ThreeGppHttpVariables::UpdateMainObjectMuAndSigma (void)
{
  NS_LOG_FUNCTION (this);
  const double a1 = std::pow (m_mainObjectSizeStdDev, 2.0);
  const double a2 = std::pow (m_mainObjectSizeMean, 2.0);
  const double a = std::log (1.0 + (a1 / a2));
  const double mu = std::log (m_mainObjectSizeMean) - (0.5 * a);
  const double sigma = std::sqrt (a);
  NS_LOG_DEBUG (this << " Mu= " << mu << " Sigma= " << sigma << ".");
  m_mainObjectSizeRng->SetAttribute ("Mu", DoubleValue (mu));
  m_mainObjectSizeRng->SetAttribute ("Sigma", DoubleValue (sigma));
}
```

3GPP HTTP服务端——嵌入对象大小

嵌入对象大小服从对数正态分布：

```
const uint32_t objectSize = m_httpVariables->GetEmbeddedObjectSize ();

ThreeGppHttpVariables::GetEmbeddedObjectSize ()
{ // 参数验证.
  if (m_embeddedObjectSizeMax <= m_embeddedObjectSizeMin)
    {NS_FATAL_ERROR ("`EmbeddedObjectSizeMax` attribute "
                     << " must be greater than"
                     << " the `EmbeddedObjectSizeMin` attribute.");  }
  //寻找一个值使它落在[min, max)区间内
  uint32_t value;
  do
  {value = m_embeddedObjectSizeRng->GetInteger ();}
  while ((value < m_embeddedObjectSizeMin) || (value >= m_embeddedObjectSizeMax))
  return value;}

m_embeddedObjectSizeRng = CreateObject<LogNormalRandomVariable> ();
```

3GPP HTTP服务端——嵌入对象大小

对数正态分布参数（均值、方差）与代码中涉及的最大最小值的设置：

```
.AddAttribute ("EmbeddedObjectSizeMean",
               "The mean of embedded object sizes (in bytes).",
               UintegerValue (7758),
               MakeUintegerAccessor (&ThreeGppHttpVariables::SetEmbeddedObjectSizeMean),
               MakeUintegerChecker<uint32_t> ())
.AddAttribute ("EmbeddedObjectSizeStdDev",
               "The standard deviation of embedded object sizes (in bytes).",
               UintegerValue (126168),
               MakeUintegerAccessor (&ThreeGppHttpVariables::SetEmbeddedObjectSizeStdDev),
               MakeUintegerChecker<uint32_t> ())
.AddAttribute ("EmbeddedObjectSizeMin",
               "The minimum value of embedded object sizes (in bytes).",
               UintegerValue (50),
               MakeUintegerAccessor (&ThreeGppHttpVariables::m_embeddedObjectSizeMin),
               MakeUintegerChecker<uint32_t> (22))
.AddAttribute ("EmbeddedObjectSizeMax",
               "The maximum value of embedded object sizes (in bytes).",
               UintegerValue (2000000), // 2 MB
               MakeUintegerAccessor (&ThreeGppHttpVariables::m_embeddedObjectSizeMax),
               MakeUintegerChecker<uint32_t> ())
```

3GPP HTTP服务端——嵌入对象大小

对数正态分布参数（均值、方差）设置：

```
void
ThreeGppHttpVariables::SetEmbeddedObjectSizeMean (uint32_t mean)
{NS_LOG_FUNCTION (this << mean);
  NS_ASSERT_MSG (mean > 0, "Mean must be greater than zero.");
  m_embeddedObjectSizeMean = mean;
  if (IsInitialized ())
    {  UpdateEmbeddedObjectMuAndSigma ();}}
void
ThreeGppHttpVariables::SetEmbeddedObjectSizeStdDev (uint32_t stdDev)
{NS_LOG_FUNCTION (this << stdDev);
  m_embeddedObjectSizeStdDev = stdDev;
  if (IsInitialized ())
    {UpdateEmbeddedObjectMuAndSigma (); }}
```

3GPP HTTP服务端——嵌入对象大小

对数正态分布参数（均值、方差）更新：

```
void
ThreeGppHttpVariables::UpdateEmbeddedObjectMuAndSigma (void)
{
  NS_LOG_FUNCTION (this);
  const double a1 = std::pow (m_embeddedObjectSizeStdDev, 2.0);
  const double a2 = std::pow (m_embeddedObjectSizeMean, 2.0);
  const double a = std::log (1.0 + (a1 / a2));
  const double mu = std::log (m_embeddedObjectSizeMean) - (0.5 * a);
  const double sigma = std::sqrt (a);
  NS_LOG_DEBUG (this << " Mu= " << mu << " Sigma= " << sigma << ".");
  m_embeddedObjectSizeRng->SetAttribute ("Mu", DoubleValue (mu));
  m_embeddedObjectSizeRng->SetAttribute ("Sigma", DoubleValue (sigma));
}
```

模型用法

- ThreeGppHttpServerHelper
- ThreeGppHttpClientHelper
- ThreeGppHttpServer
- ThreeGppHttpClient
- ThreeGppHttpVariables

- 使用ThreeGppHttpServerHelper和ThreeGppHttpClientHelper，实现在节点上安装ThreeGppHttpServer和ThreeGppHttpClient应用。
- Helper对象可以对客户端和服务器对象进行属性配置，但无法对ThreeGppHttpVariables对象进行配置。
- ThreeGppHttpVariables的配置通过直接修改其属性来实现
- HTTP应用的运行例子：$ ./waf --run 'three-gpp-http-example'，该例子输出客户端的网页请求和服务器端的响应。

HTTP业务

- HTTP应用的运行例子：$ ./waf --run 'three-gpp-http-example'，输出客户端的网页请求和服务器端的响应。

```
woo@woo-virtual-machine:~/workspace/ns-allinone-3.32/ns-3.32$ ./waf --run three-
gpp-http-example
Waf: Entering directory `/home/woo/workspace/ns-allinone-3.32/ns-3.32/build'
[2705/2759] Compiling scratch/three-gpp-http-example.cc
[2706/2759] Compiling scratch/subdir/scratch-simulator-subdir.cc
[2707/2759] Compiling scratch/scratch-simulator.cc
[2708/2759] Compiling scratch/myfirst.cc
[2717/2759] Linking build/scratch/subdir/subdir
[2718/2759] Linking build/scratch/scratch-simulator
[2719/2759] Linking build/scratch/myfirst
[2720/2759] Linking build/scratch/three-gpp-http-example
Waf: Leaving directory `/home/woo/workspace/ns-allinone-3.32/ns-3.32/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (7.290s)
+0.006272000s Client has established a connection to the server.
+0.006918400s Server generated a main object of 79431 bytes.
+0.006918400s Server sent a packet of 79453 bytes.
+0.009948800s Client received a packet of 536 bytes from 04:07:0a:01:01:02:50:00
```