The APPLPy User's Guide

Matthew Robinson

October 1, 2014

Contents

1 Introduction

Welcome to APPLPy and the world of open-source computational probability! This guides is intended to serve as a reference for new users. The topics it will cover include installation, options for running APPLPy, and a basic description of each procedure. Before exploring these topics, however, we will begin with a brief overview of the APPLPy project and what it hopes to accomplish for its end users.

1.1 What is APPLPy?

APPLPy stands for A Probability Programming Language – Python Edition. The primary goal of APPLPy is to provide an open-source conceptual probability package capable of manipulating random variables symbolically. Although python development has only occurred recently, it is derived from Maple-based project known as A Probability Programming Language (APPL). Since its creation, APPL has been successfully integrated into both undergraduate and graduate level courses at the College of William and Mary and the United States Military Academy, while also facilitating research in areas ranging from order statistics to queuing theory. The hope of APPLPy is to make the computational capabilities of APPL available to researchers and educators on an open-source platform.

1.2 How is APPLPy used?

Although APPLPy can be used for a variety of purposes, it is best suited to fill three special roles. First, it enables students to gain an intuitive understanding of mathematical statistics by automating tedious, calculus-intensive algorithms. As such, students can experiment with different models without having to perform difficult derivations or produce ad-hoc code. Second, it allows students to check hand derived results. This aids the learning process by providing a quick and reliable 'answer key'. Finally, it allows researchers to explore systems whose properties would be intractable to derive by hand. As mentioned above, the Maple-based APPL software has already spawned a variety of insightful research. APPLPy has the potential to continue along this pathway.

2 Installation

This section will cover the installation of the APPLPy software, as well as its dependencies.

2.1 Dependencies

APPLPy is coded in Python 2 and requires Python 2.6 or later. The software has not yet been tested for forward compatibility with Python 3. In addition to python, APPLPy requires the following software packages:

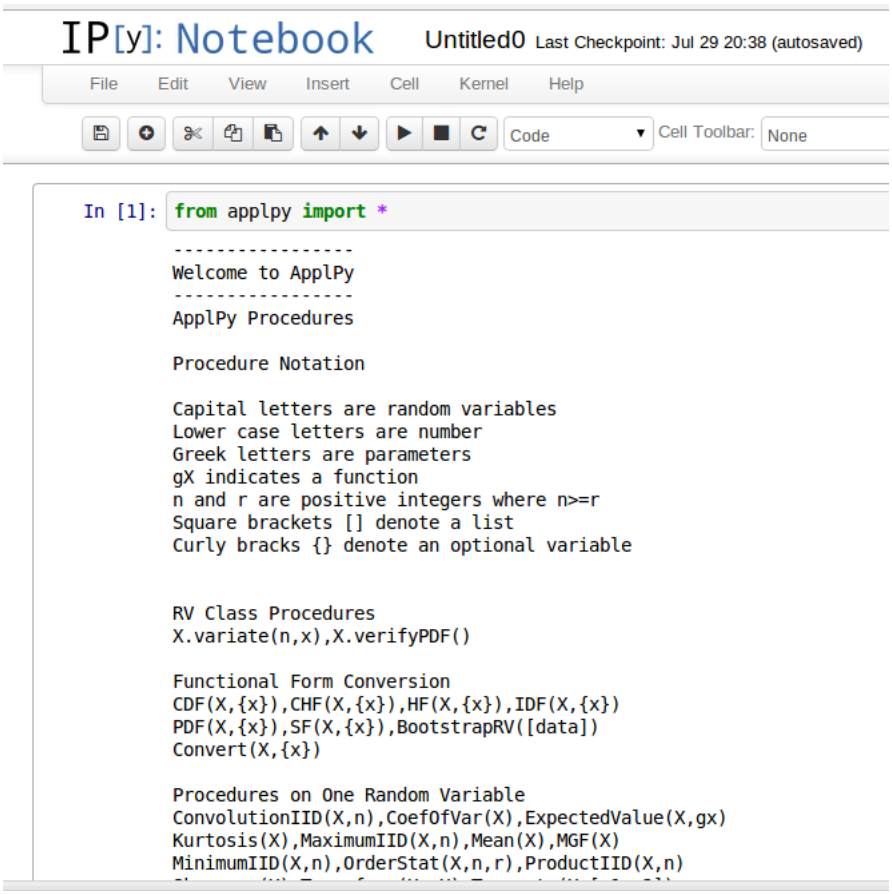
1. SymPy
2. Matplotlib

SymPy provides computer algebra capabilities and Matplotlib enables plotting. Both packages can by downloaded from the Python Package Index at https://pypi.python.org/pypi/. To install them, open the command prompt or terminal and navigate to the folder that contains the downloaded package. Once there, type the command `python setup.py install`. After they are installed, both packages will be available from any interactive python session. Alternatively, users may download the Anaconda python distribution, which ships with SymPy, Matplotlib and a number of other scientific computing tools pre-installed. The Anaconda python distribution is highly recommended and can be downloaded at https://store.continuum.io/cshop/anaconda/.

2.2 APPLPy

Download The latest version of APPLPy is v0.2.0, which was released on 17 September 2014. It is available for download from the Python Package Index at https://pypi.python.org/pypi/APPLPy/0.2.0. Users who want the latest development version of APPLPy can also download source code from https://github.com/MthwRobinson/APPLPy. To install APPLPy, navigate to the location of the downloaded files and type `python setup.py install` into the terminal or the command prompt. After the installation is complete, users can run APPLPy from any python interactive session by typing `from applpy import *`.

Options for running APPLPy APPLPy can be run from any python interactive session. For fast performance, users can run APPLPy from the command prompt or from the interactive IDLE interface that ships with every python distribution. Another option is the iPython Notebook, which executes procedures more slowly, but provides a notebook interface that is similar to Maple or Mathematica. APPLPy sessions that are launched from the iPython Notebook can be saved for future use. A screen shot of an APPLPy session running in iPyhon Notebook is shown below.



Portability Since APPLPy has been developed entirely in Python, it is compatible with almost any operating system. APPLPy will run happily on Window, Linux or Mac.

3 Procedures

It is now time to begin exploring APPLPy's capabilities. Each APPLPy procedure currently falls into one of five categories: procedures relating to the random variable class, procedures for changing the functional form of a random variable, procedures on one random variable, procedures on two random variables and utility procedures. For the remainder of the guide, the following notation will hold:

- Capital letters X Y denote random variables
- Lower case letter x y denote variates
- Greek letters θ κ denote parameters

- n and r denote integers where $n \geq r$
- gX is a function
- Square brackets [ ] denote a list
- Curly brackets { } denote an optional parameter

3.1 The Random Variable Class

3.1.1 Random Variable Class

Syntax RV([func],[support],{[ftype]})

Description The Random Variable class forms the core data structure of APPLPy, and underlies APPLPy's ability to process piecewise distributions. The first argument in the initialization procedure is a list of functions. The second is a list of support values. For continuous distributions, this will contain one more element than the function list, because each function requires two endpoints. For discrete distributions, both lists will contain the same number of elements. The final argument is a list consisting of two string. One indicates whether the distribution is discrete or continuous, and the other determines the functional representation of the random variable. If the third argument is not entered, the initialization procedure assumes that the user has input a continuous PDF. All string in the ftype list are entered in all lower case letters. Possible values for the functional representation string are cdf, chf, hf, idf, pdf and sf. The following code shows how to initialize a continuous piecewise PDF and a discrete piecewise PDF.

- `[♯ Create a continuous, piecewise pdf`
- `[In] X=RV([x-1,3-x],[1,2,3]['continuous','pdf'])`
- `♯ Create a discrete, piecewise pdf`
- `[In] Y=RV([1/4,1/4,1/4,1/4],[1,2,3,4],['discrete','pdf'])`

3.1.2 Special Random Variables

Syntax *DistName*RV(θ_1,θ_2,...,θ_n)

Description APPLPy comes has pre-coded a number of commonly used distribution in order to make the process of generating distributions more streamlined. Examples include the Exponential, Triangular and Normal distributions. These procedures are called by replace *DistName* in the syntax above with the name of the distribution in question. Each distribution is coded as a sub-class of the main random variable class. In distributions whose parameters have restrictions, the relevant assumptions have been built into the sub-class. For instance, the ExponentialRV class assumes that the parameter is a positive number. If the distribution is called with no arguments, then a distribution with unspecified parameters is returned. The following sytax shows how to generate a fully specified, partially specified, and unspecified Weibull distribution. Note that lambda cannot be used as a parameter, because `lambda` is a predefined Python term that creates an in line procedure. For a full list of distribution types, type `Menu()` in an APPLPy session. Also note that rational inputs must be in the form of a SymPy rational number. Python will automatically evaluate any input that is given using the division operator. For instance, entering the command `ExponentialRV(1/3)` will result in the creation of an exponential random variable with a parameter equal to 0.333333. The correct syntax to give a rational number to the value of the parameter is `ExponentialRV(Rational(1,3))`.

- `♯ Create an unspecified Weibull distribution`
- `[In] X=WeibullRV()`
- `♯ Create a partially specified Weibull distribution`
- `[In] X=WeibullRV(kappa=Rational(1,2))`
- `♯ Create a fully specified Weibull distribution`
- `[In] X=WeibullRV(Rational(1,2),2)`

3.1.3 Variate Generation

Syntax X.variate({n},{s})

Description The random variable class contains an algorithm designed to produce random variates from a given distribution. For the standard random variable class, this is accomplished using numerical methods. Specifically, the Newton-Raphson method is used to find the CDF value that corresponds to a randomly generated quantile. The optional parameter n determines the number

of parameters to be drawn. If left unspecified, n defaults to 1. The optional parameter s determines the quantile of the variate. if left unspecified, a random quantile is used. For each of the commonly used distributions, the inverse CDF has been hard coded in order to facilitate faster variate generation. As such, random variate generation is generally faster for pre-defined distributions. The following code shows how lists of random variates can be generated in APPLPy.

- `♯ Create a random variable`
- `[In] X=TriangularRV(1,2,3)`
- `♯ Create a list of 5 random variates`
- `[In] X.variate(5)`
- `[Out] [1.40918,1.619644,1.946498,2.27927,2.441339]`
- `♯ Find the 75th percentile`
- `[In] X.variate(s=.75)[0]`
- `[Out] 2.292893`

For this example, the floating point numbers have been truncated. By default, APPLPy will produce 32-bit or 64-bit floating point numbers, depending on the available hardware and software.

3.1.4 PDF Verification

Syntax X.verifyPDF()

Description APPLPy has a procedure that will verify whether or not a PDF is valid. This procedure takes no arguments, and returns a print statement that confirms or denies that validity of the PDF.

- `[In] X=ExponentialRV(1/2)`
- `[In] X.verifyPDF()`
- `[Out] Now checking for area ...`
- ` The area under f(x) is:  1`
- ` The pdf of the random variable:`
- ` is valid`

3.1.5 Pprint Display

Syntax X.display()

Descritption This procedure displays the random variable as a piecewise function using SymPy's pprint displayhook.

- [In] X=ExponentialRV()
- [In] X.display()
- [Out] continuous pdf with support [0,oo]:
- $$\begin{cases} \theta e^{-\theta x} & \text{for } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

3.1.6 Latex Output

Syntax X.latex()

Description This code produces the latex input for the distribution in question.

- `[In] X=TriangularRV(1,2,3)`
- `[In] X.latex()`

The latex from the output for this statement produces the code for:

$$\begin{cases} x-1 & \text{for } 1 \leq x \leq 2 \\ -x+3 & \text{for } 2 \leq x \leq 3 \\ 0 & \text{otherwise} \end{cases}$$

3.2 Functional Form Conversion

APPLPy's functional form conversion procedures alter and track the representation of random variables. Currently, the APPLPy supports six different random variable representations: cumulative distribution function ('cdf'), cumulative hazard function ('chf'), hazard function ('hf'), inverse density function ('idf'), probability density function ('pdf') and survivor function ('sf'). This section also covers special functions designed to manipulate discrete random variables.

3.2.1 Functional Form Conversion

Syntax {*CDF,CHF,HF,IDF,PDF,SF*}(X,{x})

Description The CDF,CHF,HF,IDF,PDF and SF procedures change the functional form of a random variable, and optionally evaluate it at a given point x. The syntax for each functional form is the same. When the procedure is called with just the random variable argument, the output is another random variable. If an optional point x is given, then the output is either numerical or symbolic. In cases in which the functional form is already correct, the random variable passed into the procedure is simply returned unchanged. These procedures accept both discrete and continuous random variables. The CDF and SF procedures are also useful for finding left and right tail probabilities for a given distribution at a given quantile.

- `♯ Create a random variable`
- `[In] X=TriangularRV(2,4,6)`
- `♯ Convert to hazard function form`
- `[In] HF(X).display()`
- `[Out] continuous hf with support [2,4,6]`
- $$\begin{cases} \frac{-2x+4}{x^2-4x-4} & \text{for } x \geq 2 \\ \frac{2}{-x+6} & \text{for } x \geq 4 \\ 0 & \text{otherwise} \end{cases}$$
- `♯ Find` $P(X \leq \frac{5}{2})$
- `[In] CDF(X,Rational(5,2))`
- `[Out]` $\frac{1}{32}$

3.2.2 Bootstrapping

Syntax BootstrapRV([data])

Description This code performs an infinite bootstrap of a data set. That is to say, each data point is assigned a probability of $\frac{1}{n}$, which would result if the data set were bootstrapped an infinite number of times. The code will bootstrap both symbolic and numeric data points. The numerical data points are arranged in ascending order and the symbolic data points are arranged in alphabetical order. If a data point is observed n times in a data set, it's probability value is given n times the weight of a data point that appears once.

- `♯ Bootstrap a data set`
- `data=[1,x,3,y]`
- `[In] Xstar=BootstrapRV(data)`
- `[In] Xstar.display()`
- `[Out] discrete pdf where {x->f(x)}:`
- `{1->0.25},{3->0.25},{x->0.25},{y->0.25}`

3.2.3 Convert

Syntax Convert(X,{gX})

Description This procedures transforms discrete random variables with a functional representation into discrete random variables with a floating point representation. Conversions of this type allow for discrete random variables with functional representations to be used as input in some of the procedures described below. The optional parameter gX describes the interval between values in the discrete random variable. The default is $x_{n+1} = x_n + 1$. Note that functional discrete random variables have the type maker 'Discrete' as opposed to 'discrete'. They have not yet been fully integrated into the software, but will be in future releases. Convert only works on random variables with finite support.

- `[In] X=BinomialRV(3,1/2)`
- `[In] X.display()`
- `[Out] Discrete pdf with support [0,3]:`
- $\left[6\frac{\frac{1}{2}^{x}\frac{1}{2}^{-x+3}}{x!(-x+3)!}\right]$

- `[In] Y=Convert(X)`
- `[In] Y.display()`
- `[Out] discrete pdf where {x->f(x)}:`
- `{0 -> 0.125}, {1 -> 0.375}, {2 -> 0.375}, {3 -> 0.125}`

3.3 Procedures on One Random Variable

This section describe symbolic statistical techniques that are performed on one random variable. The three major types of procedures described in this section are iterative random variable algebra procedures, expected value procedures and transformation procedures.

3.3.1 Convolution IID

Syntax ConvolutionIID(X,n)

Description This procedure computes the sum of n independent and identically distributed distributions. The algorithm works by iteratively applying the convolution procedure described below. Note that, due to the complexity of the integration involved, the convolution algorithm may take a long time to compute a solution for large values of n, or may fail to produce a solution altogether.

- `[In] X=ExponentialRV()`
- `[In] Y=ConvolutionIID(X,4)`
- `[In] Y.display()`
- `[Out] continuous pdf with support [0,oo]:`
- $$\begin{cases} \frac{1}{6}\theta^4 x^3 e^{-\theta x} & \text{for } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

3.3.2 Coefficient of Variation

Syntax CoefOfVar(X)

Description This procedure computes the coefficient of variation $\frac{\sigma}{\mu}$ of the random variable.

- `[In] X=WeibullRV()`
- `[In] CoefOfVar(X)`
- `[Out]`

$$\frac{\theta^{-\kappa\left(1-\frac{\kappa+1}{\kappa}\right)}\sqrt{-\theta^{2\kappa\left(1-\frac{\kappa+1}{\kappa}\right)}\Gamma^2\left(1+\frac{1}{\kappa}\right)+\theta^{\kappa\left(1-\frac{\kappa+2}{\kappa}\right)}\Gamma\left(1+\frac{2}{\kappa}\right)}}{\Gamma\left(1+\frac{1}{\kappa}\right)}$$

3.3.3 Expected Values

Syntax ExpectedValue(X,gX)

Description This procedures computes the expected value of a random variable X where X is transformed by the function g(x). The expected value procedure is used in the algorithms to find the mean, variance, skewness and kurtosis of random variables.

- `[In] X=ChiRV()`
- `[In] ExpectedValue(X,x**2)`
- `[Out]`

$$\frac{2^{\frac{1}{2}N}2^{-\frac{1}{2}N+1}\Gamma\left(\frac{1}{2}N+1\right)}{\Gamma\left(\frac{1}{2}N\right)}$$

3.3.4 Kurtosis

Syntax Kurtosis(X)

Description This procedure computes the Kurtosis of a random variable

- `[In] X=BetaRV(2,2)`
- `[In] data=X.variate(10)`
- `[In] Xstar=BootstrapRV(data)`
- `[In] Kurtosis(Xstar)`
- `[Out] 2.06567188978`

3.3.5 Maximum IID

Syntax MaximumIID(X,n)

Description This procedure finds the distribution of the maximum of n independent and identically distributed distributions by applying the Maximum algorithm iteratively. For more details concerning the maximum procedure, see its listing below.

- `[In] X=TriangularRV(1,3,7)`
- `[In] Y=MaximumIID(X,3)`
- `[Out] continuous cdf with support [1,3,7]:`
- $$\begin{cases} -\frac{1}{1728}\left(\left(-x^2+2x+11\right)^2+144\right)\left(-x^2+2x+11\right) & \text{for } 1 \le x \le 3 \\ \frac{x^2}{24}+\frac{7x}{12}-\frac{49}{24} & \text{for } 3 \le x \le 7 \\ 0 & \text{otherwise} \end{cases}$$

3.3.6 Mean

Syntax Mean(X)

Description This procedure computes the mean of the given random variable.

- `[In] X=ExponentialRV()`
- `[In] Mean(X)`
- `[Out]` $\frac{1}{\theta}$

3.3.7 Moment Generating Function

Syntax MGF(X)

Description Produces the moment generating function of a random variable X. The moment generating function is represented as a function of t.

- `[In] X=UniformRV()`
- `[In] MGF(X)`
- `[Out]` $-\frac{e^{at}}{t(-a+b)}+\frac{e^{bt}}{t(-a+b)}$

3.3.8 Minimum IID

Syntax MinimumIID(X,n)

Description This procedure finds the distribution of the minimum of n independent and identically distributed distributions by applying the Minimum algorithm iteratively. For more details concerning the maximum procedure, see its listing below.

- `[In] X=GammaRV(.5,2)`
- `[In] Y=MinimumIID(X,3)`
- `[In] Y.display()`
- `[Out] continuous cdf with support [0,oo]:`
- $$\begin{cases} \left(-\left(0.5x+1.0\right)^2+e^{1.0x}\right)e^{-1.0x} & \text{for } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

3.3.9 Order Statistics

Syntax OrderStat(X,n,r)

Description The order statistic procedure computes the distribution of the r^{th} sample drawn from the random variable X, given a sample size of n, where $n \geq r$.

- `[In] X=TriangularRV(1,5,9)`
- `[In] Y=OrderStat(X,5,2)`
- `[In] Y.display()`
- `[Out] continuous pdf with support [1,5,9]:`
- $$\begin{cases} \frac{5}{4194304}\left(x-1\right)\left(-x^2+2x+31\right)^3\left(x^2-2x+1\right) & \text{for } 1 \leq x \leq 5 \\ \frac{5}{4194304}\left(x-9\right)\left(x^2-18x+49\right)\left(x^2-18x+81\right)^3 & \text{for } 5 \leq x \leq 9 \\ 0 & \text{otherwise} \end{cases}$$

3.3.10 Product IID

Syntax ProductIID(X,n)

Description This procedure finds the distribution of the product of n independent and identically distributed distributions by applying the product algorithm iteratively. For more details concerning the product procedure, see its listing below. Note that, due to the complexity of the integration involved, the convolution algorithm may take a long time to compute a solution for large values of n, or may fail to produce a solution altogether.

- `[In] Xstar=BootstrapRV([x,y,z])`
- `[In] Ystar=ProductIID(Xstar,3)`
- `[In] Ystar.display()`
- `[Out] discrete pdf where x->f(x):`
- `{x**4 -> 0.012345}, {y**4 -> 0.012345}, {z**4 -> 0.012345}, {x*y**3 -> 0.049382}, {x*z**3 -> 0.049382}, {x**3*y -> 0.049382}, {y*z**3 -> 0.0493827}, {x**3*z -> 0.0493827}, {y**3*z -> 0.0493827]}, {x**2*y**2 -> 0.074074}, {x**2*z**2 -> 0.074074}, {y**2*z**2 -> 0.074074}, {x*y*z**2 -> 0.1481481}, {x*y**2*z -> 0.1481481}, {x**2*y*z -> 0.1481481}`

3.3.11 Skewness

Syntax Skewness(X)

Description This procedure computes the skewness of a random variable.

- `[In] X=BetaRV(2,3)`
- `[In] Skewness(X)`
- `[Out] 2/7`

3.3.12 Transform

Syntax Transform(x,gX)

Description This algorithm is used to transform a random variable X by a function gX. The transformation gX is entered using a list-of-lists data structure, where the first list of the transformation functions and the second is a support list for the transformation. Several special issues dictate how the user should input the transformation. The first is that the transformation functions

must be monotonic along each segment of the transformation. If they are not (for example in the case of $x \to x^2$), local maxima and minima must be included in the support of the transformation. For instance, $g(x) = x^2$ for $-\infty$ to ∞ would be input as [[x**2,x**2],[-oo,0,oo]]. The minimum point at zero is included in the support. User must also be sure to include any non-differentiable points in the support of the transformation. The support of the transformation may 'overshoot' the support of the random variable, as the algorithm will automatically disregard any segment that does not apply.

- `[In] X=TriangularRV(2,4,5)`
- `[In] gX=[[x**2,x**2],[-oo,0,oo]]`
- `[In] Y=Transform(X,gX)`
- `[In] Y.display()`
- `[Out] continuous pdf with support [4,16,25]:`
- $$\begin{cases} \frac{\sqrt{x}-2}{6\sqrt{x}} & \text{for } 4 \leq x \leq 16 \\ \frac{-\sqrt{x}+5}{3\sqrt{x}} & \text{for } 16 \leq x \leq 25 \\ 0 & \text{otherwise} \end{cases}$$

3.3.13 Truncation

Syntax Truncate(X,[lower,upper])

Description Truncate reduces the support of a random variable, and the renormalizes it to make it a valid PDF. The result is a distribution with the same shape as the parent distribution between the two given end points.

- `[In] X=BetaRV(2,2)`
- `[In] Y=Truncate(X,[1/4,3/4])`
- `[In] Y.display()`
- `[Out] continuous pdf with support [0.25,0.75]:`
- $$\begin{cases} \frac{96}{11} x\left(-x+1\right) & \text{for } 0.25 \leq x \leq 0.75 \\ 0 & \text{otherwise} \end{cases}$$

3.3.14 Variance

Syntax Variance(X)

Description This procedure computes the variance of a random variable.

- `[In] X=UniformRV()`
- `[In] Variance(X)`
- `[Out]` $\frac{1}{12}a^2 - \frac{1}{6}ab + \frac{1}{12}b^2$

3.4 Procedures on Two Random Variables

This section describes statistical techniques that are applied to two or more random variables. The procedures in this section generally involve random variable algebra, and form the core of APPLPy's random variable modelling capabilities.

3.4.1 Convolution

Syntax Convolution(X,Y) or X+Y

Description The convolution procedures allows for the summation of two random variables. Syntactically, this can be accomplish by either calling the Convolution procedure directly, or using the '+' operator. In the random variable class, the '+' operator is programmed to recognize convolutions as summations. Moreover, the convolution procedure can also perform subtraction by applying the transformation $x \to -x$ to the second random variable. As such, the convolution algorithm comprises part of APPLPy's core modelling capabilities. For lifetime distributions, convolutions are computed directly. However, for all other distributions, the convolution is computed by first finding the product of e^X and e^Y, and then transforming the result. The addition of a pure convolution algorithm is currently being explored.

- `[In] X=UniformRV(1,2)`
- `[In] Y=UniformRV(3,4)`
- `[In] Z=X+Y`
- `[Out] continuous pdf with support [4,5,6]`

- $\begin{cases} x-4 & \text{for } 4 \leq x \leq 5 \\ -x+6 & \text{for } 5 \leq x \leq 6 \\ 0 & \text{otherwise} \end{cases}$

3.4.2 Maximum

Syntax Maximum(X,Y)

Description This procedure computes the distribution of the maximum of two random variable.

- `[In] X=TriangularRV(2,4,6)`
- `[In] Y=TriangularRV(3,5,7)`
- `[In] Z=Maximum(X,Y)`
- `[In] Z.display()`
- `[Out] continuous pdf with support [3,4,5,6,7]:`
- $\begin{cases} \frac{x^2}{8} - \frac{3x}{4} + \frac{1}{8} & \text{for } 3 \leq x \leq 4 \\ \frac{x^2}{8} - \frac{3x}{4} + \frac{1}{8} & \text{for } 4 \leq x \leq 5 \\ -\frac{x^2}{8} + \frac{7x}{4} - \frac{49}{8} & \text{for } 5 \leq x \leq 6 \\ -\frac{x^2}{8} + \frac{7x}{4} - \frac{49}{8} & \text{for } 6 \leq x \leq 7 \\ 0 & \text{otherwise} \end{cases}$

3.4.3 Minimum

Syntax Minimum(X,Y)

Description This procedure computes the distribution of the minimum of two random variables

- `[In] X=TriangularRV(2,4,6)`
- `[In] Y=TriangularRV(3,5,7)`
- `[In] Z=Minimum(X,Y)`
- `[In] Z.display()`

- `[Out] continuous pdf with support [2,3,4,5,6]:`
- $$\begin{cases} \frac{x^2}{8} - \frac{x}{2} + \frac{1}{2} & \text{for } 2 \leq x \leq 3 \\ \frac{1}{64}\left(-x^2 + 4x + 4\right)\left(x^2 - 6x + 1\right) + 1 & \text{for } 3 \leq x \leq 4 \\ -\frac{x^2}{8} + \frac{3x}{2} - \frac{7}{2} & \text{for } 4 \leq x \leq 5 \\ -\frac{1}{64}\left(x^2 - 14x + 49\right)\left(x^2 - 12x + 36\right) + 1 & \text{for } 5 \leq x \leq 6 \\ 0 & \text{otherwise} \end{cases}$$

3.4.4 Mixture

Syntax Mixture([p1,p2,...,pn],[X1,X2,...,Xn])

Description Mixture computes the 'mixture' of a list of n random variables. The first argument is a list of weights for the random variables involve. These must sum to one. The second argument is a list of the random variable to be mixed. The distribution is a result of these weighted mixtures.

- `[In] X1=TriangularRV(2,4,6)`
- `[In] X2=TriangularRV(3,5,7)`
- `[In] X3=TriangularRV(1,5,9)`
- `[In] Y=Mixture([1/4,1/4,1/2],[X1,X2,X3])`
- `[In] Y.display()`
- `[Out] continuous pdf with support [1,2,3,4,5,6,7,9]:`
- $$\begin{cases} \frac{x}{32} - \frac{1}{32} & \text{for } 1 \leq x \leq 2 \\ \frac{3x}{32} - \frac{5}{32} & \text{for } 2 \leq x \leq 3 \\ \frac{5x}{32} - \frac{11}{32} & \text{for } 3 \leq x \leq 4 \\ \frac{x}{32} + \frac{5}{32} & \text{for } 4 \leq x \leq 5 \\ -\frac{5x}{32} + \frac{35}{32} & \text{for } 5 \leq x \leq 6 \\ -\frac{3x}{32} + \frac{23}{32} & \text{for } 6 \leq x \leq 7 \\ -\frac{x}{32} + \frac{9}{32} & \text{for } 7 \leq x \leq 9 \\ 0 & \text{otherwise} \end{cases}$$

3.4.5 Product

Syntax Product(X,Y) or X*Y

Description The product procedure computes the product of two random variables. This procedure can either be called directly, or called using the '*' operator in the random variable class. The product procedure is also used to perform random variable division using the '/' operator. This is accomplished by first transforming the second argument by the transformation $x \to \frac{1}{x}$. As such, product serves as one of APPLPy's core random variable algebra procedures.

- `[In] X=UniformRV(2,4)`
- `[In] Y=UniformRV(3,5)`
- `[In] Z=X*Y`
- `[In] Z.display()`
- `[Out] continuous pdf with support [6,10,12,20]:`
- $$\begin{cases} \frac{1}{4}\log\left(\frac{1}{3}x\right) - \frac{1}{4}\log\left(2\right) & \text{for } 6 \leq x \leq 10 \\ -\frac{1}{4}\log\left(\frac{1}{5}x\right) + \frac{1}{4}\log\left(\frac{1}{3}x\right) & \text{for } 10 \leq x \leq 12 \\ -\frac{1}{4}\log\left(\frac{1}{5}x\right) + \frac{1}{2}\log\left(2\right) & \text{for } 12 \leq x \leq 20 \\ 0 & \text{otherwise} \end{cases}$$

3.5 Statistics Procedures

APPLPy statistics procedures enable users to estimate parameters given a random variable model, as well as compute a variety of test statistics. These numerical procedures do not leverage APPLPy's symbolic capabilities in the same way as the procedures on the random variable class. They are available in most statistics packages and are included for convenience.

3.5.1 Kolmogorov-Smirnoff Test

Syntax KSTest(X,[data])

Description The KSTest procedure returns the Kolmogorov-Smirnoff test statistic, given a data set and a random variable model. In this procedure, the random variable, X, must be fully specified.

- `[In] X=NormalRV(2,2)`
- `[In] data=X.variate(n=30)`
- `[In] KSTest(X,data)`
- `[Out] 0.995045406342999`

3.5.2 Maximum Likelihood Estimates

Syntax MLE(X,[data],[parameters],[censor])

Description The MLE procedure returns the maximum likelihood estimate for a list of variables given a data set. The censor variable is a binary list corresponding to the data set where 1 indicates an observed value and 0 indicates a right censored variable.

- `[In] theta=Symbol('theta')`
- `[In] X=NormalRV(0,theta)`
- `[In] data=NormalRV(0,2).variate(n=20)`
- `[In] MLE(X,data,[theta])`
- `[Out] 1.94837832828`

3.5.3 Method of Moments

Syntax MOM(X,[data],[parameters])

Description The MOM procedures returns estimates for a list of unknown parameters using the method of moments. For a random variable model with n unknown parameters, the n^{th} moment of the random variable model is set equal to the n^{th} moment of the data set. These equations are then solved to produce parameter estimates.

- `[In] theta=Symbol('theta')`
- `[In] X=UniformRV(0,theta)`

- `[In] data=UniformRV(0,10).variate(n=20)`
- `[In] MOM(X,data,[theta])`
- `[Out]` θ : `10.93127659162`

3.6 Utility Procedures

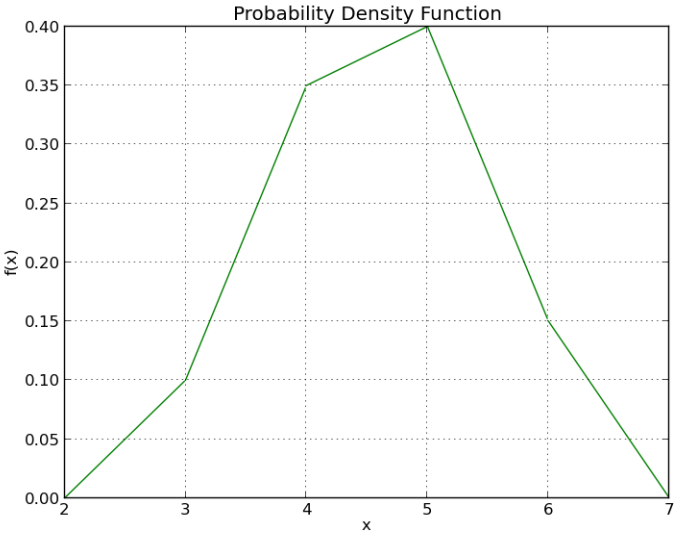
The utility procedures in APPLPy are designed to make it easier for users to process and visualize random variables. As such, the procedures in this section largely relating the the plotting of random variables.

3.6.1 Plotting Distributions

Syntax PlotDist(X,{[lower,upper]},{color},{display})

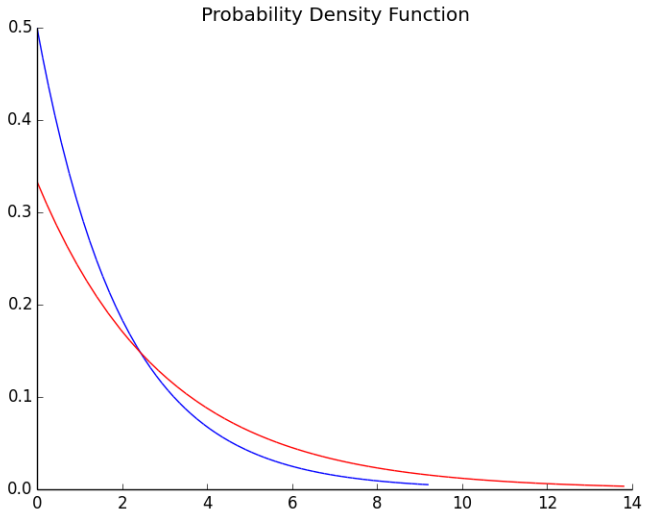
Description PlotDist is called to display a two-dimensional plot of a random variable. The first argument is the random variable to be plotted, and the second argument is an optional argument that defines the limits of the plots. If not limits are given, the supports of the random variable are used as the default. For distributions with infinite supports, the 1^{st} and 99^{th} percentile are used instead. The color arguments takes a string as input and allows the user to specify the color of the plot. Display is a boolean variable that determines whether the plot will be displayed immediately. The default value is `True`, meaning that the plot will be display when the command is executed. If display is set to `False`, the plot will wait to be displayed until prompted by the user.

- `[In] X=TriangularRV(2,4,6)`
- `[In] Y=TriangularRV(3,5,7)`
- `[In] Z=Mixture([.4,.6],[X,Y])`
- `[In] PlotDist(Z)`
- `[Out]`



Multiple plots can be displayed by following the pattern of the following commands:

- `[In] X=ExponentialRV(Rational(1,3))`
- `[In] Y=ExponentialRV(Rational(1,2))`
- `[In] xplot=PlotDist(X,color='red',display=False)`
- `[In] yplot=PlotDist(Y,color='blue',display=False)`
- `[In] totalplot=xplot.append(yplot[0])`
- `[In] totalplot.show()`
- `[Out]`



3.7 Bayesian Procedures

APPLPy includes a number of procedures designed to aid with Bayesian parameter estimation. These procedures emphasize the computation of exact distributions for parameters given a distribution with one unknown parameter, a prior distribution for the unknown parameter and a data set.

3.7.1 Credible Sets

Syntax CredibleSet(P,alpha)

Description Given a posterior distribution for a unknown parameter, the CredibleSet procedures computes a credible set for the unknown parameter, where α is the confidence level. The procedures produces the credible set as a list, where the first number is the $\frac{\alpha}{2}$ percentile of the posterior distribution and the second number is the $1 - \frac{\alpha}{2}$ percentile of the posterior distribution.

- `[In] theta=Symbol('theta')`
- `[In] X=BinomialRV(12,theta)`
- `[In] Y=TriangularRV(0,1/2,1)`
- `[In] data=[5]`
- `[In] P=Posterior(X,Y,data,theta)`

- `[In] CredibleSet(P,.05)`
- `[Out] [0.2252385, 0.6689076]`

3.7.2 Posterior

Syntax Posterior(X,Y,[data],param)

Description The posterior procedure computes the posterior distribution for an unknown parameter given a likelihood function, a prior distribution, a data set and an unknown parameter. When the posterior distribution is produced as output, the distribution will display the unknown parameter as x. This enables the random variable to remain compatible with the remainder of APPLPy, whose symbolic procedures assume that distributions are a function of x.

- `[In] theta=Symbol('theta')`
- `[In] X=BinomialRV(12,theta)`
- `[In] Y=TriangularRV(0,1/2,1)`
- `[In] data=[5]`
- `[In] P=Posterior(X,Y,data,theta)`
- `[Out]` $\begin{cases} \frac{-36900864}{1363}\theta^6(\theta-1)^7 & \texttt{for } \leq x \leq \frac{1}{2} \\ \frac{36900864}{1363}\theta^5(\theta-1)^8 & \texttt{for } \frac{1}{2} \leq x \leq 1 \\ 0 & \texttt{otherwise} \end{cases}$