WebPPL is a feature-rich probabilistic programming language embedded in Javascript.

Check out some **demos** or try it yourself in the editor below.

```
print("================================================================================")
print("PCM20201214_TriangleMeanPrior&RiskCalculation              *** 2020/12/14 *** ")
print("  see also Simple Reaction Time, Example 9, Card, Moran & Newell, 1983, p.66  ")
print("  see also https://www.humanbenchmark.com/tests/reactiontime/statistics       ")
print("  here we use the triangular distribution as a prior distribution             ")
print("  see also https://en.wikipedia.org/wiki/Triangular_distribution              ")
print("  CMN-interval 'typical[fast ~ slow]' is interpreted ...                      ")
print("            as  triangle(fast=a, slow=b, 'typical'=mean=c)                    ")
print("================================================================================")
/**
 * @author - Claus Moebus   <claus.moebus@uol.de>
 */
//------------------------------------------------------------------------------------------
/**
 * @variable {number} startTime - used in method 'runtime' to compute runtime in sec and min
 */
var startTime = Date.now()
//------------------------------------------------------------------------------------------
print("Input parameter:")
/**
 * @variable {integer} nTrials - no of efficient samples (incl. burnout) in MCMC-sampling
 */
var nTrials = 6E4
print("nTrials = " + nTrials)
//------------------------------------------------------------------------------------------
/**
 * @variable {integer} nSigma - no of standard deviations between mean and 'slow', 'fast'
 *                              interval boundaries
 */
var nSigma = 3
print("nSigma = " + nSigma)
//------------------------------------------------------------------------------------------
/**
 * @variable {integer} myBurnPeriod - length of burnin period in MCMC process
 */
var myBurnPeriod = nTrials * 0.10
print("length of burn-in period = " + myBurnPeriod)
//------------------------------------------------------------------------------------------
/**
 * @variable {integer} myLag - only every myLag-th sample will be retained during MCMC
 */
var myLag = 10
print("length of lag = " + myLag)
//------------------------------------------------------------------------------------------
/**
 * @variable {array} data - author's reaction times in an experiment found here
```

```
 *                          https://www.humanbenchmark.com/tests/reactiontime/
 *                          visited March 2018
 */
var data =
    [458, 292, 228, 403, 271, 420, 350, 235, 260, 306]
print("response time data = [" + data + "]")
print("mean of data  = " + listMean(data))
print("stdev of data = " + listStdev(data))
print("-------------------------------------------------------------------------")
/**
 * @function seqOfThresholds - generates an array of thresholds between min and max
 * @property {number}  min - minimum = fastman's value
 * @poperty  {number}  max - maximum = slowman's value
 */
var seqOfThresholds = function(min, max) {
  var range = max - min
  var stepSize =  range/50
  var increment = function(x) {x * stepSize + min}
  mapN(increment, Math.floor(range/stepSize + 1))
}
//-------------------------------------------------------------------------------------------
/**
 * @variable {array} tauPCrit   - critical values-at-risk in msec for tauP
 * @variable {array} tauCCrit   - critical values-at-risk in msec for tauC
 * @variable {array} tauMCrit   - critical values-at-risk in msec for tauM
 * @variable {array} tauSumCrit - critical values-at-risk in msec for tauSum
 */
var tauPCrit   = seqOfThresholds(100, 200) // from typical value upto slowmans value
var tauCCrit   = seqOfThresholds( 70, 170) // from typical value upto slowmans value
var tauMCrit   = seqOfThresholds( 70, 100) // from typical value upto slowmans value
var tauSumCrit = seqOfThresholds(240, 470) // from typical value upto slowmans value
print("-------------------------------------------------------------------------")
/**
 * @description - function hyperParmTauX returns the parameter c=mode fom input parameters
 *                       - 'typical'=mean, fast=a, and slow=b are taken from MHP
 *                       -  returns mode=c
 * @ function hyperParmTauX
 * @param   {number} 'typical' - value is the mean value of the CMN-interval
 * @param   {number}         a - value is the 'fast' parameter of Triangle(a, b, c)
 * @param   {number}         b - value is the 'slow' parameter of Triangle(a, b, c)
 * @returns {number}         c - value is the mode c parameter of Triangle(a, b, c)
 */
var hyperParmTauX = function(mean, a, b) {
  var c = 3*mean - (a + b)
  return{c:c, a:a, b:b}
}
var hyperParmTauP = hyperParmTauX(100.0, 50.0, 200.0)
print("hyperParmTauP = {c:" + hyperParmTauP.c + ", a:" + hyperParmTauP.a +
      ", b:" + hyperParmTauP.b + "}")
var hyperParmTauC = hyperParmTauX(70.0, 25.0, 170.0)
print("hyperParmTauC = {c: " + hyperParmTauC.c + ", a:" + hyperParmTauC.a +
      ", b:" + hyperParmTauC.b + "}")
var hyperParmTauM = hyperParmTauX(70.0, 30.0, 100.0)
print("hyperParmTauM = {c: " + hyperParmTauM.c + ", a:" + hyperParmTauM.a +
      ", b:" + hyperParmTauM.b + "}")
print("-------------------------------------------------------------------------")
/**
 * @object hyperParmSigmaTauSum - shape=a and scale=b for variance of Gaussian Likelihood
 * @property {number} a - value is the shape parameter of Gamma(a, b)
 * @property {number} b - value is the scale parameter of Gamma(a, b)
 */
```

```
var hyperParmSigmaTauSum = {a:4.0, b:20.0}
print("hyperParmSigmaTauSum = {a:" + hyperParmSigmaTauSum.a + ", b:" + hyperParmSigmaTauSum.b
print("-------------------------------------------------------------------------------")
//-----------------------------------------------------------------------------------------
// function definitions
//-----------------------------------------------------------------------------------------
/**
 * @function runtime - method to compute the runtime in seconds and minutes
 */
var runTime = function() {
  var stopTime = Date.now()
  var runSecs = (stopTime - startTime)/1000
  var runMins = runSecs/60
  print("runtime in seconds = " + runSecs)
  print("runtime in minutes = " + runMins)}
//-----------------------------------------------------------------------------------------
/**
 * @description - descriptive statistics of a sample-generated distribution
 * @function myTauXDistribution
 * @param {string} id - The identifier of the tauX distribution.
 * @param {distributionObject} tauXDistribution - tauX distribution (X = P, C, M, T)
 * @param {number} modeTauX - mode of tauX as a function of a and b
 *                            mode = (a-1)*b for a >= 1
 * @returns {object} meanSigmaTauObject - object with mean and sigma of TauX
 * @property {number} meanTauX - mean of tauX (X = P, C, M, T) or tau
 * @property {number} sigmaTauX - standard deviation of tauX (X = P, C, M, T) or tau
 */
var myTauXDescription = function(id, tauXDistribution, modeTauX) {
  var myTauXDistribution = { // extraction of probs and support from WebPPL tauX distributior
    probs: map(function(eventTuple){ // object to compute mean and sigma of tauX
      Math.exp(tauXDistribution.score(eventTuple))}, tauXDistribution.support()),
    support: tauXDistribution.support()}
  print(id)
  // mode(tauX), mean(tauX), variance(tauX) and sigma(tauX)
  print("mode = " + modeTauX)
  var meanTauX = sum(map2(function(value, prob) {
    value*prob},myTauXDistribution.support, myTauXDistribution.probs))
  print("mean = " + meanTauX)
  var sigmaTauX = Math.sqrt(sum(map2(function(value, prob) {
    Math.pow((value-meanTauX), 2)*prob},
                                    myTauXDistribution.support,
                                    myTauXDistribution.probs)))
  print("sigma = " + sigmaTauX)
  var tauX_Intval = {fast:meanTauX - nSigma * sigmaTauX, mean:meanTauX,
                     slow:meanTauX + nSigma * sigmaTauX}
  return tauX_Intval}
//-----------------------------------------------------------------------------------------
/**
 * @description - cdf computes the cumulative density function P(X <= c)
 * @function cdf
 * @param {distributionObject} distrObject - must be generated by function 'Infer'
 * @param {real} c - function argument of cdf F(c) = P(X <= c)
 * @returns {real} - F(c) = P(X <= c)
 */
var cdf = function(distrObject, c) {
  var support = distrObject.support()
  var probs = map(function(xValue){
    Math.exp(distrObject.score(xValue))
  }, support)
  sum(map2(function(prob, xValue) {
    xValue <= c ? prob : 0
```

```
  }, probs, support))
}
//-------------------------------------------------------------
/**
 * @description - probsAtRisk computes the cumulative density function 1-F(c) = P(X > c)
 * @function probsAtRisk
 * @param {distributionObject} distrObject - must be generated by function 'Infer'
 * @param {real} valsAtRisk - function arguments of cdf 1-F(c) = P(X > c)
 * @returns {array} - F(c_i) = P(X <= c_i)  ; i = 1, ...
 */
var probsAtRisk = function (distrObject, valsAtRisk) {
  map(function(valAtRisk) {
    1.0 - cdf(distrObject, valAtRisk)
  }, valsAtRisk)
}
//-------------------------------------------------------------
/**
 * @ description - prints a table of two column vectors:
 *               -  values-at-risk and risk probabilities
 */
var printRiskProbs = function(valsAtRisk, valsAtRiskText, probs) {
  /*
  map2(function(valAtRisk, prob) {
    print(valsAtRiskText + " = " +  valAtRisk + ";  risk probability = " + prob)
  }, valsAtRisk, probs)
  */
}
//-------------------------------------------------------------
/**
 * @description - prints a table of two column vectors:
 *              - values-at-risk and increase of risk probabilities
 * @function displayDiffProbs
 */
var displayDiffProbs = function(valsAtRisk, valsAtRiskText, probsPrior, probsPosterior) {
  var probDiffs = map2(function(priorPr, postPr) {
    postPr - priorPr  // change
  }, probsPrior, probsPosterior)
  map2(function(valAtRisk, probDiff) {
    if (probDiff < 0.05) {print(valsAtRiskText + " = " +  valAtRisk
                                + "; increase in risk probs = " + probDiff)}
    else {/* empty */ ;}}
      , valsAtRisk, probDiffs)
  viz.line(valsAtRisk, probDiffs, {xLabel: valsAtRiskText, yLabel: "Risk Excess"})
}
//=============================================================================================
/**
 * @description         - draws one sample from the Triangle(a, b, c)-distribution
 *                      - // https://en.wikipedia.org/wiki/Triangular_distribution
 * @function            - oneSampleOfTriangle
 * @param (number) fast - is the lower bound of the CMN-interval and a of Triangle(a, b, c)
 * @param (number) slow - is the upper bound of the CMN-interval and b of Triangle(a, b, c)
 * @param (number) mode - is the mode of the CMN-interval  and param c of Triangle(a, b, c)
 */
var oneSampleOfTriangle = function(a, b, c) {
  var u  = sample(Uniform({a:0, b:1}))
  var ba = b - a
  var bc = b - c
  var ca = c - a
  var Fc = ca / ba
  var x  = (0 < u) && (u < Fc) ?
      (a + Math.sqrt(u * ba * ca)) :
```

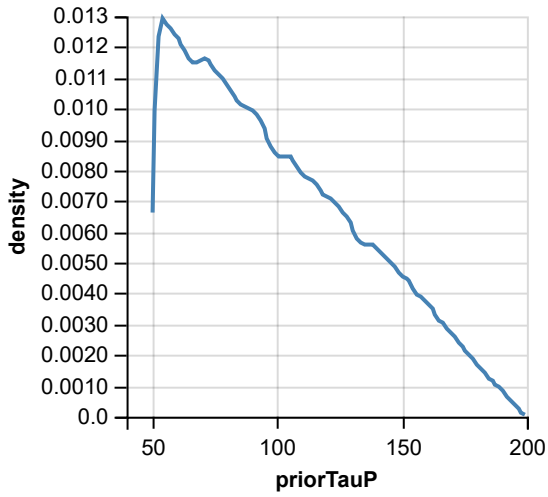
```
      (b - Math.sqrt((1 - u) * ba * bc))
  return x
}
//
//-------------------------------------------------------------------------------------
/**
 * @function oneSampleOfPriors      - takes  o n e  sample from all priors tauP, tauC, tauM,
 *                                  -  tauSum = tauP + tauC + tauM, and sigmaTauSum
 * @returns {object} sampleOfPriors - o n e  priors-tuple
 * @returns {object} priorSigmaTauSum - o n e  sample from the Gamma distr
 *                                  - this is prior sigma for the Gaussian likelihood
 */
var oneSampleOfPriors = function () {
  var priorTauP = oneSampleOfTriangle(hyperParmTauP.a,hyperParmTauP.b,hyperParmTauP.c)
  var priorTauC = oneSampleOfTriangle(hyperParmTauC.a,hyperParmTauC.b,hyperParmTauC.c)
  var priorTauM = oneSampleOfTriangle(hyperParmTauM.a,hyperParmTauM.b,hyperParmTauM.c)
  var priorTauSum = priorTauP + priorTauC + priorTauM
  var priorSigmaTauSum =
      sample(Gamma({shape:hyperParmSigmaTauSum.a, scale:hyperParmSigmaTauSum.b}))
  return {priorTauP:priorTauP, priorTauC:priorTauC, priorTauM:priorTauM,
          priorTauSum:priorTauSum, priorSigmaTauSum:priorSigmaTauSum}
}
//-------------------------------------------------------------------------------------
/**
 * @description - Infer generates an multivariate prior distribution for TauX
 * @variable {distribution} priorTauX - value is a WebPPL distribution object
 */
var priorTauX = Infer({model:oneSampleOfPriors, method: 'forward', samples: nTrials})
print('Univariate Priors TauX (X=P, C, M, Sum, SigmaTauSum) ~ Gamma(???, ???)')
viz.marginals(priorTauX)
print("-------------------------------------------------------------------------------")
print("model-generated "+ nSigma + "*sigma tau-interval: ")
var priorTauPIntval =
    myTauXDescription("priorTauP", marginalize(priorTauX,'priorTauP'), "unknown")
print("{fast:" + priorTauPIntval.fast + "  mean:" + priorTauPIntval.mean + "  slow:" + prior
var tauPProbsPrior = probsAtRisk(marginalize(priorTauX,'priorTauP'), tauPCrit)
printRiskProbs(tauPCrit, 'tauPCrit', tauPProbsPrior)
print("-------------------------------------------------------------------------------")
var priorTauCIntval =
    myTauXDescription("priorTauC", marginalize(priorTauX,'priorTauC'), "unknown")
print("{fast:" + priorTauCIntval.fast + "  mean:" + priorTauCIntval.mean + "  slow:" + prior
var tauCProbsPrior = probsAtRisk(marginalize(priorTauX,'priorTauC'), tauCCrit)
printRiskProbs(tauCCrit, 'tauCCrit', tauCProbsPrior)
print("-------------------------------------------------------------------------------")
var priorTauMIntval =
    myTauXDescription("priorTauM", marginalize(priorTauX,'priorTauM'), "unknown")
print("{fast:" + priorTauMIntval.fast + "  mean:" + priorTauMIntval.mean + "  slow:" + prior
var tauMProbsPrior = probsAtRisk(marginalize(priorTauX,'priorTauM'), tauMCrit)
printRiskProbs(tauMCrit, 'tauMCrit', tauMProbsPrior)
print("-------------------------------------------------------------------------------")
var priorTauSumIntval =
    myTauXDescription("priorTauSum", marginalize(priorTauX,'priorTauSum'), "unknown")
print("{fast:" + priorTauSumIntval.fast + "  mean:" + priorTauSumIntval.mean + "  slow:" + pr
var tauSumProbsPrior = probsAtRisk(marginalize(priorTauX,'priorTauSum'), tauSumCrit)
printRiskProbs(tauSumCrit, 'tauSumCrit', tauSumProbsPrior)
print("-------------------------------------------------------------------------------")
var priorSigmaTauSum_Intval =
    myTauXDescription("priorSigmaTauSum", marginalize(priorTauX,'priorSigmaTauSum'), "unknow
print("model-generated "+ nSigma + "*sigma tau-interval: ")
print("{fast:" + priorSigmaTauSum_Intval.fast + "  mean:" + priorSigmaTauSum_Intval.mean + "
print("===============================================================================")
```

```
/**
 * @function oneSampleOfModel - takes  o n e  sample from the priors
 * @returns {object} posteriorTauSum - returns  o n e  sample of posterior TauSum-tuple
 */
var oneSampleOfModel = function() {
  /**
   * @variable {number} PriorTauSum - a sample from Gamma TauSum-distribution
   */
  var priorTauP = oneSampleOfTriangle(hyperParmTauP.a,hyperParmTauP.b,hyperParmTauP.c)
  var priorTauC = oneSampleOfTriangle(hyperParmTauC.a,hyperParmTauC.b,hyperParmTauC.c)
  var priorTauM = oneSampleOfTriangle(hyperParmTauM.a,hyperParmTauM.b,hyperParmTauM.c)
  var priorTauSum = priorTauP + priorTauC + priorTauM
  /**
   * @variable {number} priorSigmaTauSum - a sample from SigmaTauSum Gamma distribution
   */
  var priorSigmaTauSum =
      sample(Gamma({shape:hyperParmSigmaTauSum.a, scale:hyperParmSigmaTauSum.b}))
  //
  map(function(datum) {
      observe(Gaussian({mu:priorTauSum, sigma:priorSigmaTauSum}),datum)
  }, data)
  return {postTauP: priorTauP, postTauC: priorTauC,  postTauM: priorTauM,
          postTauSum:priorTauSum, postSigmaTauSum:priorSigmaTauSum}
}
//-------------------------------------------------------------------------------
/**
 * @description - Infer generates the posterior distribution 'posteriorTauT'
 * @variable {distributionObject} posteriorTauT - univariate posterior distribution
 */
print('Univariate Posteriors TauX (X=P, C, M, Sum) Gamma(???, ???) and SigmaTauSum Gamma(???,
var posterior = Infer({model:oneSampleOfModel, method:'MCMC', samples: nTrials,
                           burn:myBurnPeriod, lag:myLag})
viz.marginals(posterior)
print("-------------------------------------------------------------------------")
print("model-generated "+ nSigma + "*sigma tau-interval: ")
var postTauPIntval =
    myTauXDescription("postTauP", marginalize(posterior,'postTauP'), "unknown")
print("{fast:" + postTauPIntval.fast + "  mean:" + postTauPIntval.mean + "  slow:" + postTaul
var tauPProbsPosterior = probsAtRisk(marginalize(posterior,'postTauP'), tauPCrit)
printRiskProbs(tauPCrit, 'tauPCrit', tauPProbsPosterior)
print("-----------------------------------------------------------")
displayDiffProbs(tauPCrit, 'tauPCrit', tauPProbsPrior, tauPProbsPosterior)
print("-------------------------------------------------------------------------")
var postTauCIntval =
    myTauXDescription("postTauC", marginalize(posterior,'postTauC'), "unknown")
print("{fast:" + postTauCIntval.fast + "  mean:" + postTauCIntval.mean + "  slow:" + postTauC
var tauCProbsPosterior = probsAtRisk(marginalize(posterior,'postTauC'), tauCCrit)
printRiskProbs(tauCCrit, 'tauCCrit', tauCProbsPosterior)
print("-----------------------------------------------------------")
displayDiffProbs(tauCCrit, 'tauCCrit', tauCProbsPrior, tauCProbsPosterior)
print("-------------------------------------------------------------------------")
var postTauMIntval =
    myTauXDescription("postTauM", marginalize(posterior,'postTauM'), "unknown")
print("{fast:" + postTauMIntval.fast + "  mean:" + postTauMIntval.mean + "  slow:" + postTauM
var tauMProbsPosterior = probsAtRisk(marginalize(posterior,'postTauM'), tauMCrit)
printRiskProbs(tauMCrit, 'tauMCrit', tauMProbsPosterior)
print("-----------------------------------------------------------")
displayDiffProbs(tauMCrit, 'tauMCrit', tauMProbsPrior, tauMProbsPosterior)
print("-------------------------------------------------------------------------")
var postTauSumIntval =
    myTauXDescription("postTauSum", marginalize(posterior,'postTauSum'), "unknown")
```

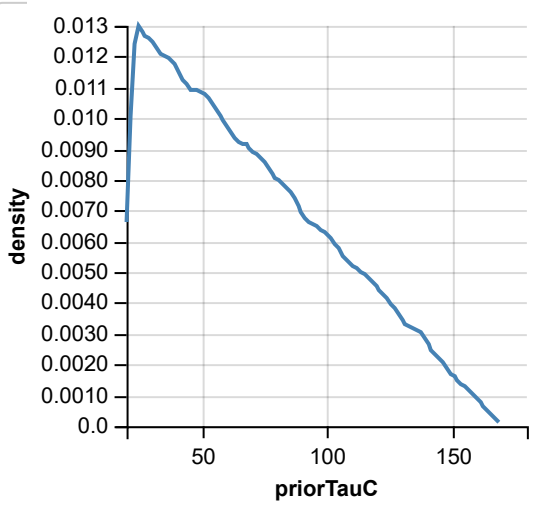
```
print("{fast:" + postTauSumIntval.fast + "  mean:" + postTauSumIntval.mean + "  slow:" + post
var tauSumProbsPosterior = probsAtRisk(marginalize(posterior,'postTauSum'), tauSumCrit)
printRiskProbs(tauSumCrit, 'tauSumCrit', tauSumProbsPosterior)
print("-------------------------------------------------------")
displayDiffProbs(tauSumCrit, 'tauSumCrit', tauSumProbsPrior, tauSumProbsPosterior)
print("-------------------------------------------------------------------------")
var postSigmaTauSumIntval =
    myTauXDescription("postSigmaTauSum", marginalize(posterior,'postSigmaTauSum'), "unknown")
print("{fast:" + postSigmaTauSumIntval.fast + "  mean:" + postSigmaTauSumIntval.mean + "  sl
print("=========================================================================")
runTime()
print("=========================================================================================")
```

run

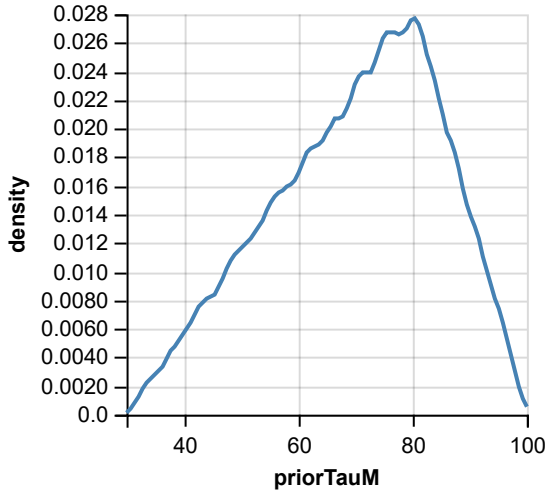
```
=========================================================================
PCM20201214_TriangleMeanPrior&RiskCalculation              *** 2020/12/14 ***
  see also Simple Reaction Time, Example 9, Card, Moran & Newell, 1983, p.66
  see also https://www.humanbenchmark.com/tests/reactiontime/statistics
  here we use the triangular distribution as a prior distribution
  see also https://en.wikipedia.org/wiki/Triangular_distribution
  CMN-interval 'typical[fast ~ slow]' is interpreted ...
            as  triangle(fast=a, slow=b, 'typical'=mean=c)
=========================================================================
Input parameter:
nTrials = 60000
nSigma = 3
length of burn-in period = 6000
length of lag = 10
response time data = [458,292,228,403,271,420,350,235,260,306]
mean of data  = 322.3
stdev of data = 77.10389095240265
-------------------------------------------------------------------------
-------------------------------------------------------------------------
hyperParmTauP = {c:50, a:50, b:200}
hyperParmTauC = {c: 15, a:25, b:170}
hyperParmTauM = {c: 80, a:30, b:100}
-------------------------------------------------------------------------
hyperParmSigmaTauSum = {a:4, b:20}
-------------------------------------------------------------------------
Univariate Priors TauX (X=P, C, M, Sum, SigmaTauSum) ~ Gamma(???, ???)
priorTauP:
```



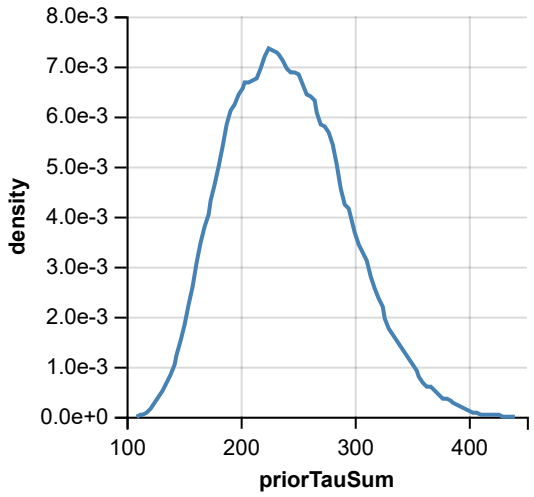
priorTauC:



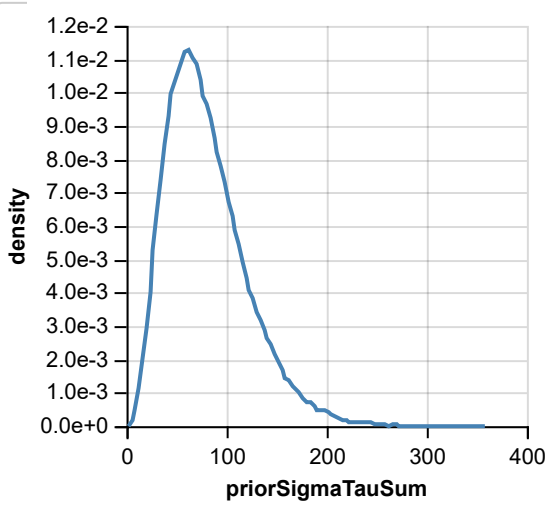
priorTauM:



priorTauSum:

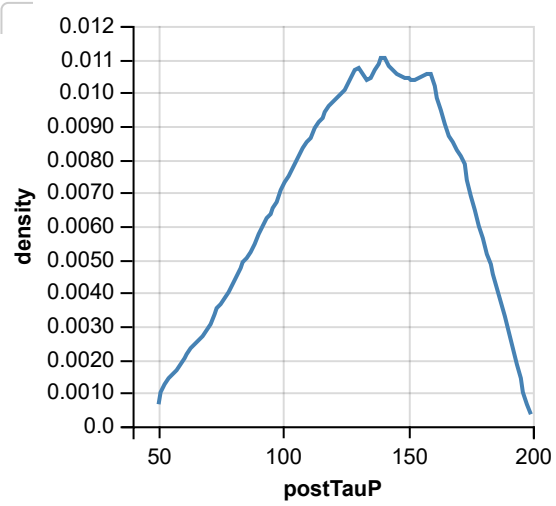


priorSigmaTauSum:

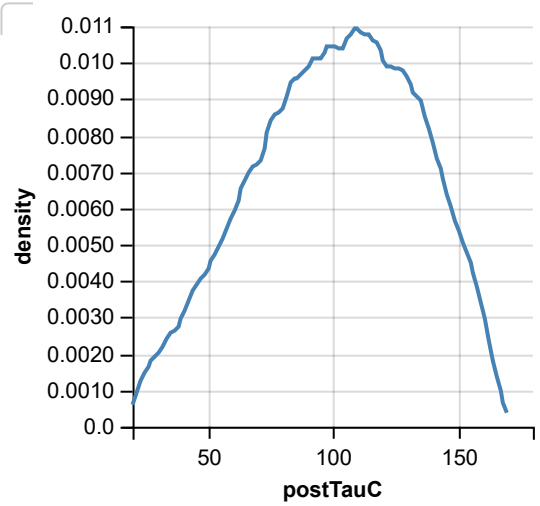



```
--------------------------------------------------------------------------------
model-generated 3*sigma tau-interval:
priorTauP
mode = unknown
mean = 100.09499907698596
sigma = 35.30654898340906
{fast:-5.824647873241219  mean:100.09499907698596  slow:206.01464602721313}
--------------------------------------------------------------------------------
priorTauC
mode = unknown
mean = 69.91611261279145
sigma = 35.27321632904632
{fast:-35.90353637434751  mean:69.91611261279145  slow:175.7357615999304}
--------------------------------------------------------------------------------
priorTauM
mode = unknown
mean = 69.97200888934576
sigma = 14.74086533719751
{fast:25.74941287775323  mean:69.97200888934576  slow:114.19460490093829}
--------------------------------------------------------------------------------
priorTauSum
mode = unknown
```

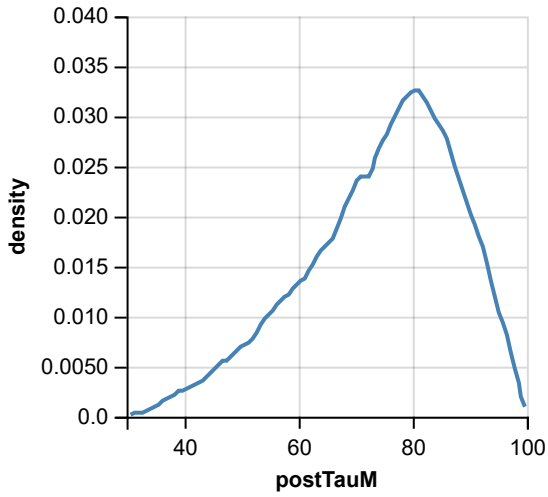
```
mean = 239.98312057912256
sigma = 52.00021503658485
{fast:83.98247546936801  mean:239.98312057912256  slow:395.9837656888771}
------------------------------------------------------------------------------
priorSigmaTauSum
mode = unknown
mean = 80.10513393251213
sigma = 40.10637209828167
model-generated 3*sigma tau-interval:
{fast:-40.21398236233287  mean:80.10513393251213  slow:200.42425022735713}
==============================================================================
Univariate Posteriors TauX (X=P, C, M, Sum) Gamma(???, ???) and SigmaTauSum Gamma
(???, ???)
postTauP:
```



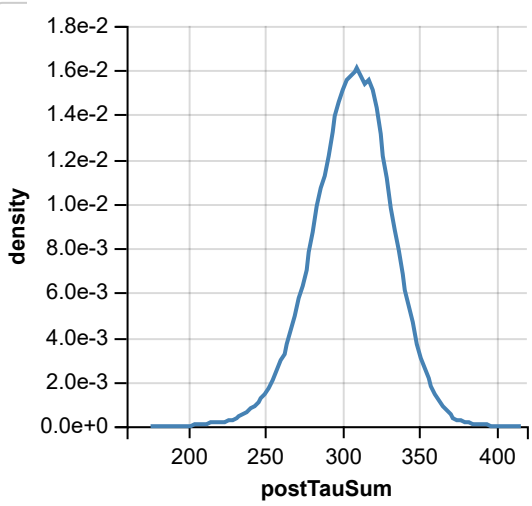

```
postTauC:
```



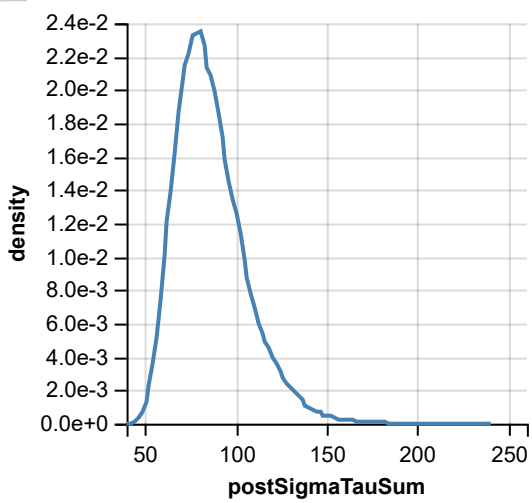

```
postTauM:
```




```
postTauSum:
```

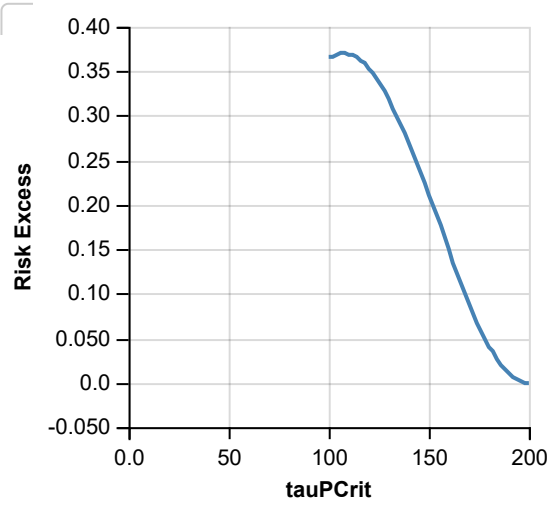



```
postSigmaTauSum:
```

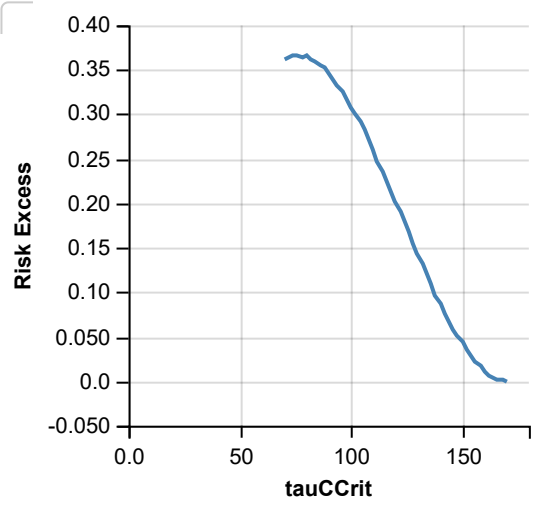



```
------------------------------------------------------------------------------
model-generated 3*sigma tau-interval:
postTauP
mode = unknown
mean = 131.3771879414163
sigma = 32.86662676470843
{fast:32.777307647291025  mean:131.3771879414163  slow:229.9770682355416}
------------------------------------------------------------
tauPCrit = 178; increase in risk probs = 0.04914999999999059
```

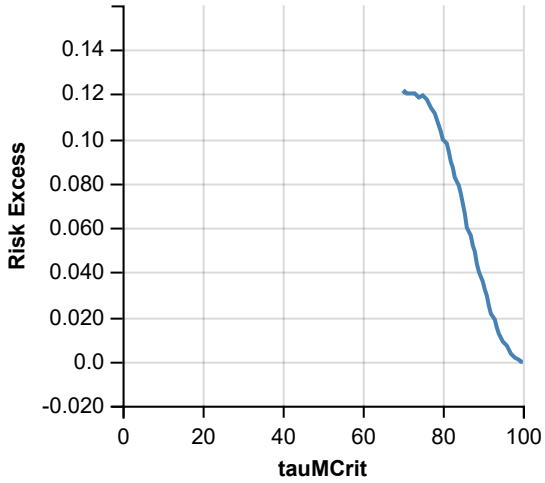
```
tauPCrit = 180; increase in risk probs = 0.04089999999999139
tauPCrit = 182; increase in risk probs = 0.03441666666665866
tauPCrit = 184; increase in risk probs = 0.026749999999992724
tauPCrit = 186; increase in risk probs = 0.019783333333326714
tauPCrit = 188; increase in risk probs = 0.014649999999993946
tauPCrit = 190; increase in risk probs = 0.010199999999994214
tauPCrit = 192; increase in risk probs = 0.006649999999994494
tauPCrit = 194; increase in risk probs = 0.003733333333328148
tauPCrit = 196; increase in risk probs = 0.0015499999999950553
tauPCrit = 198; increase in risk probs = 0.0003666666666617413
tauPCrit = 200; increase in risk probs = -4.884981308350689e-15
```




```
------------------------------------------------------------------------
postTauC
mode = unknown
mean = 100.68093537460715
sigma = 32.979313125229446
{fast:1.7429959989188148  mean:100.68093537460715  slow:199.61887475029548}
------------------------------------------------------------
tauCCrit = 150; increase in risk probs = 0.043383333333324
tauCCrit = 152; increase in risk probs = 0.035499999999991316
tauCCrit = 154; increase in risk probs = 0.02878333333332539
tauCCrit = 156; increase in risk probs = 0.021966666666659362
tauCCrit = 158; increase in risk probs = 0.01664999999999317
tauCCrit = 160; increase in risk probs = 0.010999999999993681
tauCCrit = 162; increase in risk probs = 0.007116666666660554
tauCCrit = 164; increase in risk probs = 0.003433333333327515
tauCCrit = 166; increase in risk probs = 0.0023999999999942956
tauCCrit = 168; increase in risk probs = 0.0007333333333277015
tauCCrit = 170; increase in risk probs = -5.551115123125783e-15
```

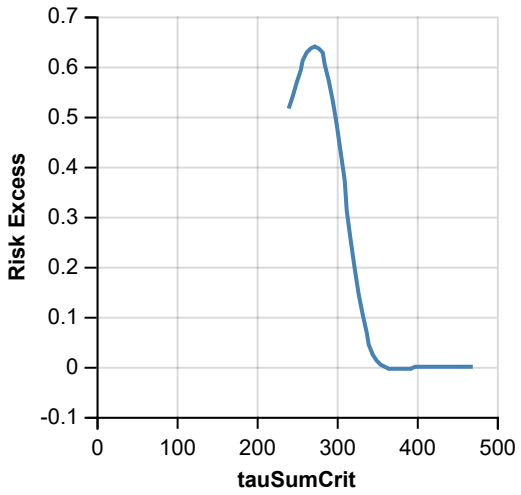



```
--------------------------------------------------------------------------------
postTauM
mode = unknown
mean = 74.2886135329192
sigma = 13.43418399435613
{fast:33.98606154985081  mean:74.2886135329192  slow:114.5911655159876}
------------------------------------------------------------
tauMCrit = 88; increase in risk probs = 0.048933333333327056
tauMCrit = 88.6; increase in risk probs = 0.04429999999999423
tauMCrit = 89.2; increase in risk probs = 0.040266666666661344
tauMCrit = 89.8; increase in risk probs = 0.03573333333332851
tauMCrit = 90.4; increase in risk probs = 0.032166666666662236
tauMCrit = 91; increase in risk probs = 0.02914999999999579
tauMCrit = 91.6; increase in risk probs = 0.02489999999999626
tauMCrit = 92.2; increase in risk probs = 0.02128333333332999
tauMCrit = 92.8; increase in risk probs = 0.018549999999996958
tauMCrit = 93.4; increase in risk probs = 0.014716666666664047
tauMCrit = 94; increase in risk probs = 0.012333333333330976
tauMCrit = 94.6; increase in risk probs = 0.010699999999997822
tauMCrit = 95.2; increase in risk probs = 0.00854999999999806
tauMCrit = 95.8; increase in risk probs = 0.006866666666664911
tauMCrit = 96.4; increase in risk probs = 0.004833333333331802
tauMCrit = 97; increase in risk probs = 0.0036999999999985933
tauMCrit = 97.6; increase in risk probs = 0.0022999999999987475
tauMCrit = 98.2; increase in risk probs = 0.00144999999999873
tauMCrit = 98.8; increase in risk probs = 0.0006999999999988127
tauMCrit = 99.4; increase in risk probs = 0.0002333333333321974
tauMCrit = 100; increase in risk probs = -1.1102230246251565e-15
```




```
--------------------------------------------------------------------------
postTauSum
mode = unknown
mean = 306.34673684894443
sigma = 25.747988692734474
{fast:229.10277077074102  mean:306.34673684894443  slow:383.59070292714785}
----------------------------------------------------------
tauSumCrit = 341.2; increase in risk probs = 0.044349999999995005
tauSumCrit = 345.8; increase in risk probs = 0.02508333333333046
tauSumCrit = 350.4; increase in risk probs = 0.012549999999998507
tauSumCrit = 355; increase in risk probs = 0.004183333333332762
tauSumCrit = 359.6; increase in risk probs = -0.0007833333333333581
tauSumCrit = 364.2; increase in risk probs = -0.003933333333333011
tauSumCrit = 368.79999999999995; increase in risk probs = -0.0048833333333329065
tauSumCrit = 373.4; increase in risk probs = -0.004966666666666231
tauSumCrit = 378; increase in risk probs = -0.004316666666666302
tauSumCrit = 382.6; increase in risk probs = -0.003833333333333022
tauSumCrit = 387.2; increase in risk probs = -0.0029999999999997806
tauSumCrit = 391.79999999999995; increase in risk probs = -0.0023333333333331874
tauSumCrit = 396.4; increase in risk probs = -0.0017166666666665886
tauSumCrit = 401; increase in risk probs = -0.001383333333333292
tauSumCrit = 405.6; increase in risk probs = -0.0008500000000000174
tauSumCrit = 410.2; increase in risk probs = -0.0006666666666667043
tauSumCrit = 414.79999999999995; increase in risk probs = -0.0005333333333333856
tauSumCrit = 419.4; increase in risk probs = -0.0003500000000000725
tauSumCrit = 424; increase in risk probs = -0.00023333333333341866
tauSumCrit = 428.6; increase in risk probs = -0.00008333333333343518
tauSumCrit = 433.2; increase in risk probs = -0.00003333333333440684
tauSumCrit = 437.79999999999995; increase in risk probs = -0.000033333333333440684
tauSumCrit = 442.4; increase in risk probs = -1.1102230246251565e-16
tauSumCrit = 447; increase in risk probs = -1.1102230246251565e-16
tauSumCrit = 451.6; increase in risk probs = -1.1102230246251565e-16
tauSumCrit = 456.2; increase in risk probs = -1.1102230246251565e-16
tauSumCrit = 460.79999999999995; increase in risk probs = -1.1102230246251565e-16
```

```
tauSumCrit = 465.4; increase in risk probs = -1.1102230246251565e-16
tauSumCrit = 470; increase in risk probs = -1.1102230246251565e-16
```




```
--------------------------------------------------------------------------------
postSigmaTauSum
mode = unknown
mean = 86.48709414896926
sigma = 19.49427675894848
{fast:28.004263872123822  mean:86.48709414896926  slow:144.9699244258147}
================================================================================
runtime in seconds = 7562.279
runtime in minutes = 126.03798333333334
================================================================================
===
```

Features

- Runs on the command line with node.js (http://nodejs.org/) or in the browser (http://docs.webppl.org/en/master/development/workflow.html#browser-version).
- Supports modular and re-usable code using packages (http://docs.webppl.org/en/master/packages.html) built on top of the npm package system, and interoperates with existing Javascript packages in the npm ecosystem.
- Includes a large and expanding library of primitive distributions. (http://docs.webppl.org/en/master/distributions.html)
- Implements a variety of inference algorithms (http://docs.webppl.org/en/master/inference/index.html), including exact inference via enumeration, rejection sampling, Sequential Monte Carlo, Markov Chain Monte Carlo, Hamiltonian Monte Carlo, and inference-as-optimization (e.g. variational inference).
- Provides inference as a first-class operator in the language, allowing for nested inference (‘inference about inference’).
- Supports optimizable models with neural network components using adnn (https://www.npmjs.com/package/adnn).

Demos

Browser-based applications powered by WebPPL.

- Procedural vines with shape constraints (demos/vines/index.html)
- 3D procedural spaceships with shape constraints (http://dritchie.github.io/web-procmod/) (Note: the code in this demo is written in an older version of WebPPL)

Local install

Install WebPPL in two easy steps:

1. Install node.js (http://nodejs.org)
2. Run `npm install -g webppl`

Now, the `webppl` command is globally available.

To upgrade to the latest version, run `npm update -g webppl`.

Documentation

To learn more about how to set up and use WebPPL, take a look at our documentation (http://docs.webppl.org) and the examples (https://github.com/probmods/webppl/tree/master/examples).

To learn more about how WebPPL works under the hood, check out our web book, The Design and Implementation of Probabilistic Programming Languages (http://dippl.org/).

For probabilistic modeling in general, our other web book, Probabilistic Models of Cognition (https://probmods.org), might be of interest.

License

The WebPPL code base is open source and freely available for commerical and non-commercial use under the MIT license (https://github.com/probmods/webppl/blob/master/LICENSE.md).

Contributions

We encourage you to contribute to WebPPL! Check out our guidelines for contributors (https://github.com/probmods/webppl/blob/master/CONTRIBUTING.md) and join the webppl-dev (https://groups.google.com/forum/#!forum/webppl-dev) mailing list.

Pronunciation

Say “web people”.

Citing

If you use WebPPL in academic projects and papers, please cite as:

> *N. D. Goodman and A. Stuhlmüller (electronic). The Design and Implementation of Probabilistic Programming Languages. Retrieved from* `http://dippl.org` *. [bibtex]*

Publications

If you publish a paper using/extending WebPPL, let us know (https://groups.google.com/forum/#!forum/webppl-dev) and we'll add it to this list:

D. Ritchie, P. Horsfall, and N. D. Goodman. Deep Amortized Inference for Probabilistic Programs (https://arxiv.org/abs/1610.05735). arXiv:1610.05735.

L. Ouyang, M. H. Tessler, D. Ly, and N. D. Goodman. Practical optimal experiment design with probabilistic programs (https://arxiv.org/abs/1608.05046). arXiv:1608.05046.

M. H. Tessler and N. D. Goodman. A Pragmatic Theory of Generic Language (https://arxiv.org/abs/1608.02926). arXiv:1608.02926.

D. Ritchie, A. Thomas, P. Hanrahan, and N. D. Goodman. Neurally-Guided Procedural Models: Amortized Inference for Procedural Graphics Programs using Neural Networks (https://arxiv.org/abs/1603.06143). NIPS 2016.

D. Ritchie, A. Stuhlmüller, and N. D. Goodman. C3: Lightweight Incrementalized MCMC for Probabilistic Programs using Continuations and Callsite Caching (https://arxiv.org/abs/1509.02151). AISTATS 2016.

M. H. Tessler and N. D. Goodman. Communicating generalizations about events (http://stanford.edu/~mtessler/papers/Tessler2016-cogsci.pdf). Proceedings of the Thirty-Eighth Annual Conference of the Cognitive Science Society, 2016.

E. J. Yoon, M. H. Tessler, N. D. Goodman, and M. C. Frank. Talking with tact: Polite language as a balance between kindness and informativity (http://stanford.edu/~mtessler/papers/YoonTessler2016-cogsci.pdf). Proceedings of the Thirty-Eighth Annual Conference of the Cognitive Science Society, 2016.

C. Graf, J. Degen, R. X. D. Hawkins, and N. D. Goodman. Animal, dog, or dalmatian? Level of abstraction in nominal referring expressions (https://cocolab.stanford.edu/papers/GrafEtAl2016-Cogsci.pdf). Proceedings of the Thirty-Eighth Annual Conference of the Cognitive Science Society, 2016.

O. Evans, A. Stuhlmüller, and N. D. Goodman. Learning the Preferences of Ignorant, Inconsistent Agents (https://stuhlmueller.org/papers/preferences-aaai2016.pdf). AAAI 2016.

A. Stuhlmüller, R. X. D. Hawkins, N. Siddharth, and N. D. Goodman. Coarse-to-Fine Sequential Monte Carlo for Probabilistic Programs (https://arxiv.org/abs/1509.02962). arXiv:1509.02962.

O. Evans, A. Stuhlmüller, and N. D. Goodman. Learning the Preferences of Bounded Agents (https://stuhlmueller.org/papers/preferences-nipsworkshop2015.pdf). Workshop on Bounded Optimality, NIPS 2015.

R. X. D. Hawkins, A. Stuhlmüller, J. Degen, and N. D. Goodman. Why do you ask? Good questions provoke informative answers (https://stuhlmueller.org/papers/qa-cogsci2015.pdf). Proceedings of the Thirty-Seventh Annual Conference of the Cognitive Science Society, 2015.

G. Scontras and M. H. Tessler (electronic). Composition in Probabilistic Language Understanding (http://gscontras.github.io/ESSLLI-2016/). Retrieved from `http://gscontras.github.io/ESSLLI-2016` *.*

O. Evans, A. Stuhlmüller, J. Salvatier, and D. Filan (electronic). Modeling Agents with Probabilistic Programs (http://agentmodels.org). Retrieved from `http://agentmodels.org` *.*

N. D. Goodman and J. B. Tenenbaum (electronic). Probabilistic Models of Cognition (http://probmods.org). Retrieved from `http://probmods.org` *.*

N. D. Goodman and A. Stuhlmüller (electronic). The Design and Implementation of Probabilistic Programming Languages (http://dippl.org). Retrieved from `http://dippl.org` *.*

Acknowledgments

The WebPPL project is supported by grants from DARPA, under agreement number FA8750-14-2-0009, and the Office of Naval Research, grant number N00014-13-1-0788.

WebPPL is a Stanford CoCoLab (http://cocolab.stanford.edu/) project