# Fit Function explorer

March 5, 2016

In [7]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib as mpl
import matplotlib.gridspec as gridspec

import numpy as np
from ipywidgets import interact, interactive, fixed
import ipywidgets as widgets
import scipy.stats

from scipy.stats import lognorm, gamma, weibull_min, alpha, invweibull
from scipy.optimize import minimize

from collections import OrderedDict
import math
from itertools import izip
from copy import deepcopy
```

In [131]:
```
mpl.rcParams['figure.figsize'] = (9.0, 5.0)  # default size of plots
mpl.rcParams['font.size'] = 16
mpl.rcParams['axes.labelsize'] = 16
mpl.rcParams['xtick.labelsize'] = 14
mpl.rcParams['ytick.labelsize'] = 14
mpl.rcParams['legend.fontsize'] = 16
```

In [9]:
```
# %config InlineBackend.figure_format='svg'
# %config InlineBackend.figure_format='retina'
```

In [10]:
```
txt_filename = ("/Users/robina/Soolin_Users_L1JEC_CMSSW_8_0_0_pre6_Local/L1Trigger/L1JetEnergy
                "Stage2_HF_QCDFlatSpring15BX25HCALFix_12Feb_85a0ccf_noJEC_fixedPUS/rsp_clean.t

with open(txt_filename) as f:
    rsp = [float(x) for x in f]
```

In [11]:
```
rsp = np.array(rsp)
rspInv = 1./rsp
```

In [12]:
```
txt_filename = ("/Users/robina/Soolin_Users_L1JEC_CMSSW_8_0_0_pre6_Local/L1Trigger/L1JetEnergy
                "Stage2_HF_QCDFlatSpring15BX25HCALFix_12Feb_85a0ccf_noJEC_fixedPUS/rsp_ptRef10

with open(txt_filename) as f:
    rspHigh = [float(x) for x in f]
```

In [13]:
```
rspHigh = np.array(rspHigh)
rspHighInv = 1./rspHigh
```

# 1 Scipy fit functions

| Function | Works on response? | Works on 1/response? |
|---|---|---|
| **Lognormal** | Kinda | Not really |
| Gamma | Kinda | No |
| Alpha | No | Kinda |
| weibull_min | No | No |
| **invweibull** | Kinda | Kinda |

Note that if we find a satisfactory fit function for response, we can easily transform into 1/response space via the Jacobian.

## 1.1 Lower ptRef bin (10 - 14 Gev)

## 1.2 response

In [185]:
```
fit_fns_small = OrderedDict()
fit_fns_small["Normal"] = dict(fn=scipy.stats.norm)
fit_fns_small["Lognormal"] = dict(fn=scipy.stats.lognorm)
fit_fns_small["Gamma"] = dict(fn=scipy.stats.gamma)
fit_fns_small["Weibull min"] = dict(fn=scipy.stats.weibull_min)
fit_fns_small["Inv. weibull"] = dict(fn=scipy.stats.invweibull)
fit_fns_small["Inv. gauss"] = dict(fn=scipy.stats.invgauss)
fit_fns_small["Fisk"] = dict(fn=scipy.stats.fisk)
fit_fns_small["Burr"] = dict(fn=scipy.stats.burr)
fit_fns_small["Inv. gamma"] = dict(fn=scipy.stats.invgamma)
fit_fns_small["Chi2"] = dict(fn=scipy.stats.chi2)
```

In [162]:
```
fit_fns =  OrderedDict()
fit_fns["Beta"] = dict(fn=scipy.stats.beta)
fit_fns["Betaprime"] = dict(fn=scipy.stats.betaprime)
fit_fns["Burr"] = dict(fn=scipy.stats.burr)
fit_fns["Chi"] = dict(fn=scipy.stats.chi)
fit_fns["Chi2"] = dict(fn=scipy.stats.chi2)
fit_fns["Exponnorm"] = dict(fn=scipy.stats.exponnorm)
fit_fns["Exponweib"] = dict(fn=scipy.stats.exponweib)
fit_fns["F"] = dict(fn=scipy.stats.f)
fit_fns["Fatiguelife"] = dict(fn=scipy.stats.fatiguelife)
fit_fns["Fisk"] = dict(fn=scipy.stats.fisk)
fit_fns["Frechet_l"] = dict(fn=scipy.stats.frechet_l)
fit_fns["Genlogistic"] = dict(fn=scipy.stats.genlogistic)
fit_fns["Genextreme"] = dict(fn=scipy.stats.genextreme)
fit_fns["Gamma"] = dict(fn=scipy.stats.gamma)
fit_fns["Gengamma"] = dict(fn=scipy.stats.gengamma)
fit_fns["Gumbel_r"] = dict(fn=scipy.stats.gumbel_r)
fit_fns["Invgamma"] = dict(fn=scipy.stats.invgamma)
fit_fns["Invgauss"] = dict(fn=scipy.stats.invgauss)
fit_fns["Invweibull"] = dict(fn=scipy.stats.invweibull)
fit_fns["Johnsonsb"] = dict(fn=scipy.stats.johnsonsb)
fit_fns["Johnsonsu"] = dict(fn=scipy.stats.johnsonsu)
```

```
        fit_fns["Kstwobign"] = dict(fn=scipy.stats.kstwobign)
        fit_fns["Lognorm"] = dict(fn=scipy.stats.lognorm)
        fit_fns["Mielke"] = dict(fn=scipy.stats.mielke)
        fit_fns["Norm"] = dict(fn=scipy.stats.norm)
        fit_fns["Pearson3"] = dict(fn=scipy.stats.pearson3)
        fit_fns["Powerlognorm"] = dict(fn=scipy.stats.powerlognorm)
        fit_fns["Rayleigh"] = dict(fn=scipy.stats.rayleigh)
        fit_fns["Rice"] = dict(fn=scipy.stats.rice)
        fit_fns["Recipinvgauss"] = dict(fn=scipy.stats.recipinvgauss)
        fit_fns["Weibull_max"] = dict(fn=scipy.stats.weibull_max)
```

```
In [15]: def get_bin_centers(bins):
             return np.array([0.5 * (bins[i]+bins[i+1]) for i in range(len(bins)-1)])
```

```
In [133]: def plot_multiple_fits(data, fit_fns, x_label, x_range, n_fit_std=10):
              """Plot multiple fits to the data, show all.

              data: numpy.array. Data to fit to.
              fit_fns: dict[name, dict]. Function to fit, and name.
              x_label: str. Label for x axis
              x_range: list[min, max]. Range of x axis
              """
              ncols = 3
              nrows = int(math.ceil(len(fit_fns)/2.))
              fig = plt.gcf()
              fig.set_size_inches(ncols * 5, nrows * 5)
              plt.subplots_adjust(hspace=0.5)

              x_val = np.linspace(x_range[0], x_range[1], 100)

              for i_plt, (fn_name, fit_fn_dict) in enumerate(fit_fns.iteritems(), 1):
                  print "Doing", fn_name
                  #          if i_plt == 2:
        #             break
                  plt.subplot(nrows, ncols, i_plt)
                  ax = plt.gca()
                  ax.set_title(fn_name + ' fit')
                  ax.set_xlabel(x_label)

                  # apply optional cut to data
                  mean = data.mean()
                  std = data.std()
                  mask = (data < mean + (std*n_fit_std)) & (data > mean-(std*n_fit_std))
                  data = data[mask]
        #         ax.set_yscale('log')

                  # plot hist
                  n, bins, patches = ax.hist(data, bins=30, range=x_range, normed=True)

                  # fit
                  try:
                      fit_results = fit_fn_dict['fn'].fit(data)
                  except NotImplementedError:
                      continue
                  print fit_results
```

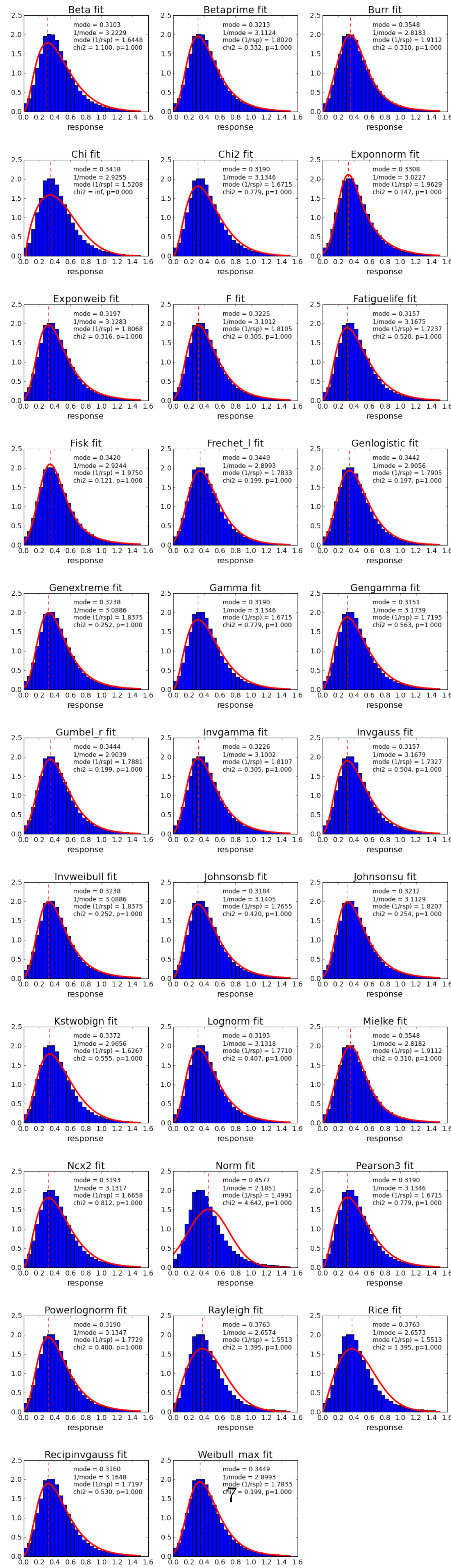
```
                has_shape_param = len(fit_results) >= 3
                loc = fit_results[-2]
                scale = fit_results[-1]
                shape = None
                if has_shape_param:
                    shape = fit_results[:-2]

                fit_fn_dict['shape'] = shape
                fit_fn_dict['loc'] = loc
                fit_fn_dict['scale'] = scale

                if has_shape_param:
                    frozen_fit = fit_fn_dict['fn'](*shape, loc=loc, scale=scale)
                else:
                    frozen_fit = fit_fn_dict['fn'](loc=loc, scale=scale)

                # get mode for fitted fn
                ave = 0.5*(x_range[0]+x_range[1])
                max_result = minimize(lambda x: -1. * frozen_fit.pdf(x), x0=ave)
                mode = max_result.x[0]

                # get mode for proper fn for (1/x) - include jacobian
                max_result_inv = minimize(lambda x: -1. * np.power(1./x, 2) * frozen_fit.pdf(1./x), x0
                mode_inv = max_result_inv.x[0]

                # do chi2 test
                bc = get_bin_centers(bins)
                predicted = np.array([frozen_fit.pdf(x) for x in bc])
                ddof = len(shape)+2 if has_shape_param else 2
                chisq, p = scipy.stats.chisquare(n, f_exp=predicted, ddof=ddof)
                fit_fn_dict['chi2'] = chisq
                fit_fn_dict['p'] = p
                print shape, loc, scale, mode, mode_inv, chisq, p

                # plot fitted fn
                y_val = frozen_fit.pdf(x_val)
                ax.plot(x_val, y_val, 'r', linewidth=3)
                ax.text(0.4, 0.65,
                        'mode = %.4f\n1/mode = %.4f\nmode (1/rsp) = %.4f\nchi2 = %.3f, p=%.3f' % (mode
                        transform=ax.transAxes, fontsize=12)

                # arrow for mode
                ax.vlines(mode, ax.get_ylim()[0], ax.get_ylim()[1], colors=['red'], linestyles='dashed
```

```
In [135]: rsp_fit_fns = deepcopy(fit_fns)
          plot_multiple_fits(rsp, rsp_fit_fns, 'response', [0, 1.5])
```

```
Doing Beta
(3.0230300333300359, 1782041379998.6787, 0.0061096755142609205, 267936968994.44775)
(3.0230300333300359, 1782041379998.6787) 0.00610967551426 267936968994.0 0.310280404338 1.64482798646 1
Doing Betaprime
(16.344245616259741, 10.569644063270601, -0.15111300234983244, 0.35620154983819408)
(16.344245616259741, 10.569644063270601) -0.15111300235 0.356201549838 0.321299454005 1.80203286093 0.3
Doing Burr
(3.7989036602894894, 0.57362014248279913, 0.0091212733978884037, 0.50023249918693202)
```

```
(3.7989036602894894, 0.57362014248279913) 0.00912127339789 0.500232499187 0.354829293068 1.91119243152 0
Doing Chi
(1.6223851355228081, 0.03092780207862468, 0.39407158959909694)
(1.6223851355228081,) 0.0309278020786 0.394071589599 0.341816344928 1.52084801998 inf 0.0
Doing Chi2
(6.6640346313361079, -0.0042681556087542311, 0.069315821308007508)
(6.6640346313361079,) -0.00426815560875 0.069315821308 0.319023374245 1.67150123797 0.778887462951 1.0
Doing Exponnorm
(2.2999810196267463, 0.21745192896381976, 0.10443653905280499)
(2.2999810196267463,) 0.217451928964 0.104436539053 0.330830955551 1.96289747214 0.14743336475 1.0
Doing Exponweib
(37.234489126932345, 0.79039233066905723, -0.20905507335695628, 0.10654136370863454)
(37.234489126932345, 0.79039233066905723) -0.209055073357 0.106541363709 0.319659756248 1.80679909955 0
Doing F
(211.66781759998321, 19.968175216793107, -0.25313033488189518, 0.639276137159972)
(211.66781759998321, 19.968175216793107) -0.253130334882 0.63927613716 0.322455042488 1.81046665951 0.30
Doing Fatiguelife
(0.42364953467310607, -0.13716766757299764, 0.54586469248275837)
(0.42364953467310607,) -0.137167667573 0.545864692483 0.315711123126 1.72374491566 0.520246482021 1.0
Doing Fisk
(3.8624814633240279, -0.086403097159995418, 0.49133657802261932)
(3.8624814633240279,) -0.08640309716 0.491336578023 0.341950119231 1.97500251507 0.120925582103 1.0
Doing Frechet_l
(7619.3425984471869, 1449.8429883650872, 1449.4980977692076)
(7619.3425984471869,) 1449.84298837 1449.49809777 0.344915458557 1.78327176662 0.199397439788 1.0
Doing Genlogistic
(541.41544304835759, -0.84665747775273892, 0.18919333120968085)
(541.41544304835759,) -0.846657477753 0.18919333121 0.344160603028 1.79050756895 0.196887317742 1.0
Doing Genextreme
(-0.073488485292052513, 0.33681848313975854, 0.18447365110757974)
(-0.073488485292052513,) 0.33681848314 0.184473651108 0.32377076065 1.83752570574 0.252467518803 1.0
Doing Gamma
(3.3319676805752261, -0.0042658391169378288, 0.1386325107522734)
(3.3319676805752261,) -0.00426583911694 0.138632510752 0.319020836187 1.67150639164 0.778931362552 1.0
Doing Gengamma
(10.151331038798961, 0.61265933451644439, -0.039804956064415531, 0.010751386667247207)
(10.151331038798961, 0.61265933451644439) -0.0398049560644 0.0107513866672 0.315073705964 1.71950778713
Doing Gumbel_r
(0.34436145822823983, 0.18959506100231005)
None 0.344361458228 0.189595061002 0.344361340456 1.78814463031 0.19902324597 1.0
Doing Invgamma
(9.9648854382754202, -0.28170934898421485, 6.6257912432156356)
(9.9648854382754202,) -0.281709348984 6.62579124322 0.32256428535 1.81068124854 0.305023767541 1.0
Doing Invgauss
(0.18120272930205344, -0.14519334638214776, 3.3269266751309829)
(0.18120272930205344,) -0.145193346382 3.32692667513 0.315670098707 1.73274879573 0.503791863564 1.0
Doing Invweibull
(13.607991237166885, -2.1735203539229007, 2.5103433682933725)
(13.607991237166885,) -2.17352035392 2.51034336829 0.323775532561 1.83750391411 0.252481931932 1.0
Doing Johnsonsb
(10.009559049722476, 2.289140507563352, -0.11800359213495037, 42.131028136357145)
(10.009559049722476, 2.289140507563352) -0.118003592135 42.1310281364 0.318422594467 1.76545017892 0.420
Doing Johnsonsu
(-2.9836657131875755, 2.0285903440025788, 0.0047586897934226271, 0.19439734630024863)
```

```
(-2.9836657131875755, 2.0285903440025788) 0.00475868979342 0.1943973463 0.321243399388 1.82067151743 0.
Doing Kstwobign
(-0.35548793220799696, 0.94183763856419789)
None -0.355487932208 0.941837638564 0.337200144411 1.62672401499 0.555184235435 1.0
Doing Lognorm
(0.42484031636902841, -0.12417582966468652, 0.53119838747982184)
(0.42484031636902841,) -0.124175829665 0.53119838748 0.319301307971 1.77096551761 0.406970029581 1.0
Doing Mielke
(2.1791220545194623, 3.7989302402685574, 0.0091211134289452232, 0.50023482409617581)
(2.1791220545194623, 3.7989302402685574) 0.00912111342895 0.500234824096 0.354830698624 1.91118995934 0
Doing Ncx2
(6.4424608340236684, 1.2241991022626382, -0.0018510243038890326, 0.059948169253531455)
(6.4424608340236684, 1.2241991022626382) -0.00185102430389 0.0599481692535 0.319319108152 1.66577965789
Doing Norm
(0.45765475257061072, 0.26428265684062685)
None 0.457654752571 0.264282656841 0.457654745345 1.49910247805 4.64240351171 0.999999563116
Doing Pearson3
(1.0956662411628391, 0.45765071897792242, 0.25305337759380087)
(1.0956662411628391,) 0.457650718978 0.253053377594 0.319019838649 1.67151707009 0.77891441464 1.0
Doing Powerlognorm
(0.78203315178967681, 0.37921621582792725, -0.14340888622753117, 0.50586019851671948)
(0.78203315178967681, 0.37921621582792725) -0.143408886228 0.505860198517 0.31901076691 1.77290847134 0
Doing Rayleigh
(0.0067427822817162468, 0.36957187601099484)
None 0.00674278228172 0.369571876011 0.376314635353 1.55129490206 1.39462159277 1.0
Doing Rice
(0.00094108302785169784, 0.0067459713859989844, 0.3695699844982272)
(0.00094108302785169784,) 0.006745971386 0.369569984498 0.376316014981 1.55129731958 1.39466440617 1.0
Doing Recipinvgauss
(0.19684433222182118, -0.12619423832387205, 0.096025475590688011)
(0.19684433222182118,) -0.126194238324 0.0960254755907 0.315974518158 1.71968473538 0.530281338625 1.0
Doing Weibull_max
(7619.3425984471869, 1449.8429883650872, 1449.4980977692076)
(7619.3425984471869,) 1449.84298837 1449.49809777 0.344915458557 1.78327176662 0.199397439788 1.0
```


Beta fit
mode = 0.3103
1/mode = 3.2229
mode (1/rsp) = 1.6448
chi2 = 1.100, p=1.000
Betaprime fit
mode = 0.3213
1/mode = 3.1124
mode (1/rsp) = 1.8020
chi2 = 0.332, p=1.000
Burr fit
mode = 0.3548
1/mode = 2.8183
mode (1/rsp) = 1.9112
chi2 = 0.310, p=1.000
Chi fit
mode = 0.3418
1/mode = 2.9255
mode (1/rsp) = 1.5208
chi2 = inf, p=0.000
Chi2 fit
mode = 0.3190
1/mode = 3.1346
mode (1/rsp) = 1.6715
chi2 = 0.779, p=1.000
Exponnorm fit
mode = 0.3308
1/mode = 3.0227
mode (1/rsp) = 1.9629
chi2 = 0.147, p=1.000
Exponweib fit
mode = 0.3197
1/mode = 3.1283
mode (1/rsp) = 1.8068
chi2 = 0.316, p=1.000
F fit
mode = 0.3225
1/mode = 3.1012
mode (1/rsp) = 1.8105
chi2 = 0.305, p=1.000
Fatiguelife fit
mode = 0.3157
1/mode = 3.1675
mode (1/rsp) = 1.7237
chi2 = 0.520, p=1.000
Fisk fit
mode = 0.3420
1/mode = 2.9244
mode (1/rsp) = 1.9750
chi2 = 0.121, p=1.000
Frechet_l fit
mode = 0.3449
1/mode = 2.8993
mode (1/rsp) = 1.7833
chi2 = 0.199, p=1.000
Genlogistic fit
mode = 0.3442
1/mode = 2.9056
mode (1/rsp) = 1.7905
chi2 = 0.197, p=1.000
Genextreme fit
mode = 0.3238
1/mode = 3.0886
mode (1/rsp) = 1.8375
chi2 = 0.252, p=1.000
Gamma fit
mode = 0.3190
1/mode = 3.1346
mode (1/rsp) = 1.6715
chi2 = 0.779, p=1.000
Gengamma fit
mode = 0.3151
1/mode = 3.1739
mode (1/rsp) = 1.7195
chi2 = 0.563, p=1.000
Gumbel_r fit
mode = 0.3444
1/mode = 2.9039
mode (1/rsp) = 1.7881
chi2 = 0.199, p=1.000
Invgamma fit
mode = 0.3226
1/mode = 3.1002
mode (1/rsp) = 1.8107
chi2 = 0.305, p=1.000
Invgauss fit
mode = 0.3157
1/mode = 3.1679
mode (1/rsp) = 1.7327
chi2 = 0.504, p=1.000
Invweibull fit
mode = 0.3238
1/mode = 3.0886
mode (1/rsp) = 1.8375
chi2 = 0.252, p=1.000
Johnsonsb fit
mode = 0.3184
1/mode = 3.1405
mode (1/rsp) = 1.7655
chi2 = 0.420, p=1.000
Johnsonsu fit
mode = 0.3212
1/mode = 3.1129
mode (1/rsp) = 1.8207
chi2 = 0.254, p=1.000
Kstwobign fit
mode = 0.3372
1/mode = 2.9656
mode (1/rsp) = 1.6267
chi2 = 0.555, p=1.000
Lognorm fit
mode = 0.3193
1/mode = 3.1318
mode (1/rsp) = 1.7710
chi2 = 0.407, p=1.000
Mielke fit
mode = 0.3548
1/mode = 2.8182
mode (1/rsp) = 1.9112
chi2 = 0.310, p=1.000
Ncx2 fit
mode = 0.3193
1/mode = 3.1317
mode (1/rsp) = 1.6658
chi2 = 0.812, p=1.000
Norm fit
mode = 0.4577
1/mode = 2.1851
mode (1/rsp) = 1.4991
chi2 = 4.642, p=1.000
Pearson3 fit
mode = 0.3190
1/mode = 3.1346
mode (1/rsp) = 1.6715
chi2 = 0.779, p=1.000
Powerlognorm fit
mode = 0.3190
1/mode = 3.1347
mode (1/rsp) = 1.7729
chi2 = 0.400, p=1.000
Rayleigh fit
mode = 0.3763
1/mode = 2.6574
mode (1/rsp) = 1.5513
chi2 = 1.395, p=1.000
Rice fit
mode = 0.3763
1/mode = 2.6573
mode (1/rsp) = 1.5513
chi2 = 1.395, p=1.000
Recipinvgauss fit
mode = 0.3160
1/mode = 3.1648
mode (1/rsp) = 1.7197
chi2 = 0.530, p=1.000
Weibull_max fit
mode = 0.3449
1/mode = 2.8993
mode (1/rsp) = 1.7833
chi2 = 0.199, p=1.000
response

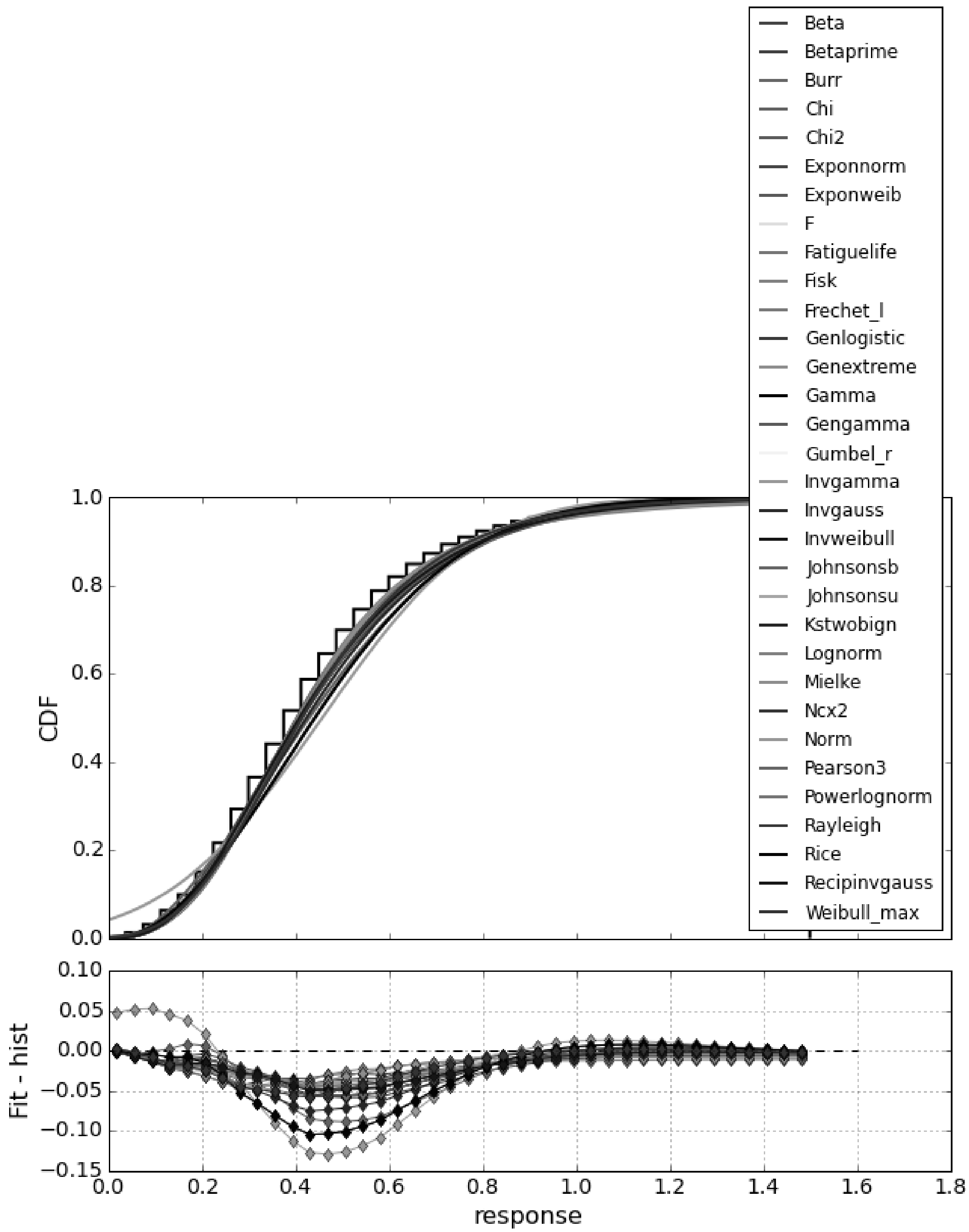
```
In [158]: def print_ordered_fit_fn(d):
              tmp = OrderedDict(sorted(d.items(), key=lambda t: t[1]))
              for k, v in tmp.iteritems():
                  print k, v['chi2'], v['p']

In [161]: print_ordered_fit_fn(rsp_fit_fns)

Fisk 0.120925582103 1.0
Exponnorm 0.14743336475 1.0
Genlogistic 0.196887317742 1.0
Gumbel_r 0.19902324597 1.0
Frechet_l 0.199397439788 1.0
Weibull_max 0.199397439788 1.0
Genextreme 0.252467518803 1.0
Invweibull 0.252481931932 1.0
Johnsonsu 0.254125123818 1.0
F 0.30451952672 1.0
Invgamma 0.305023767541 1.0
Mielke 0.309717386419 1.0
Burr 0.309724260204 1.0
Exponweib 0.31593073437 1.0
Betaprime 0.332241953068 1.0
Powerlognorm 0.399528464997 1.0
Lognorm 0.406970029581 1.0
Johnsonsb 0.42032826053 1.0
Invgauss 0.503791863564 1.0
Fatiguelife 0.520246482021 1.0
Recipinvgauss 0.530281338625 1.0
Kstwobign 0.555184235435 1.0
Gengamma 0.563359218603 1.0
Chi2 0.778887462951 1.0
Pearson3 0.77891441464 1.0
Gamma 0.778931362552 1.0
Ncx2 0.812333320214 1.0
Beta 1.0995904788 1.0
Rayleigh 1.39462159277 1.0
Rice 1.39466440617 1.0
Norm 4.64240351171 0.999999563116
Chi inf 0.0

In [86]: def calc_hist_fn_diff(n, bins, fn):
             centers = get_bin_centers(bins)
             fn_vals = np.array([fn(x) for x in centers])
             return fn_vals - n

In [173]: def plot_cdf(data, fit_fns, x_label, x_range):
              """Plot CDF for data compared with fit_fns. Also draws residuals plot."""
              fig = plt.figure()
              fig.set_size_inches(10, 8)
              gs = gridspec.GridSpec(2, 1, height_ratios=[2.2, 1])
              gs.update(hspace=0.1)
              ax1 = fig.add_subplot(gs[0])
```

```
        n, bins, _ = ax1.hist(data, normed=True, cumulative=True, bins=40,
                              range=x_range, histtype='step', color='black', linewidth=2)
        bin_centers = get_bin_centers(bins)

        x = np.linspace(x_range[0], x_range[1], 100)
        colors = np.random.rand(len(fit_fns))
        #     colors = ['red', 'dodgerblue', 'blue', 'orange',
    #               'fuchsia', 'mediumpurple', 'springgreen', 'forestgreen']
    #     colors = ['red'] * len(fit_fns)
    #     if len(colors) < len(fit_fns):
    #         new_colors = list(np.random.rand(len(fit_fns) - len(colors)))
    #         colors.extend(list(new_colors))
        diff_vals = []
        for color, (fn_name, fit_fn_dict) in izip(colors, fit_fns.iteritems()):
            loc=fit_fn_dict['loc']
            scale=fit_fn_dict['scale']
            if fit_fn_dict['shape']:
                fn_freeze = fit_fn_dict['fn'](*fit_fn_dict['shape'], loc=loc, scale=scale)
            else:
                fn_freeze = fit_fn_dict['fn'](loc=loc, scale=scale)
            y_vals = fn_freeze.cdf(x)
            diff_vals.append(calc_hist_fn_diff(n, bins, fn_freeze.cdf))
            ax1.plot(x, y_vals, color=str(color), linewidth=2, label=fn_name)
        ax1.legend(loc=4, fontsize=12)
        ax1.set_ylabel('CDF')

        ax2 = fig.add_subplot(gs[1], sharex=ax1)
        for color, diff in izip(colors, diff_vals):
            ax2.plot(bin_centers, diff, 'd-', color=str(color))
        ax2.set_xlabel(x_label)
        ax2.set_ylabel('Fit - hist')
        ax2.hlines(0, ax2.get_xlim()[0], ax2.get_xlim()[1], linestyle='dashed')
        ax2.grid(which='both')
        plt.setp(ax1.get_xticklabels(), visible=False)
```

```
In [ ]:
```

```
In [174]: plot_cdf(rsp, rsp_fit_fns, 'response', [0, 1.5])
```

```
[ 0.26169506  0.25274978  0.39884843  0.38178063  0.35650234  0.27112222
  0.35003231  0.87167893  0.45688562  0.50035705  0.47573703  0.22412766
  0.55275479  0.00935267  0.35347721  0.9522607   0.59640977  0.16277093
  0.07694122  0.39779454  0.65009455  0.14509091  0.50175399  0.55397652
  0.20488701  0.59362515  0.39942779  0.45767112  0.21738848  0.01735793
  0.08705855  0.20174117]
```

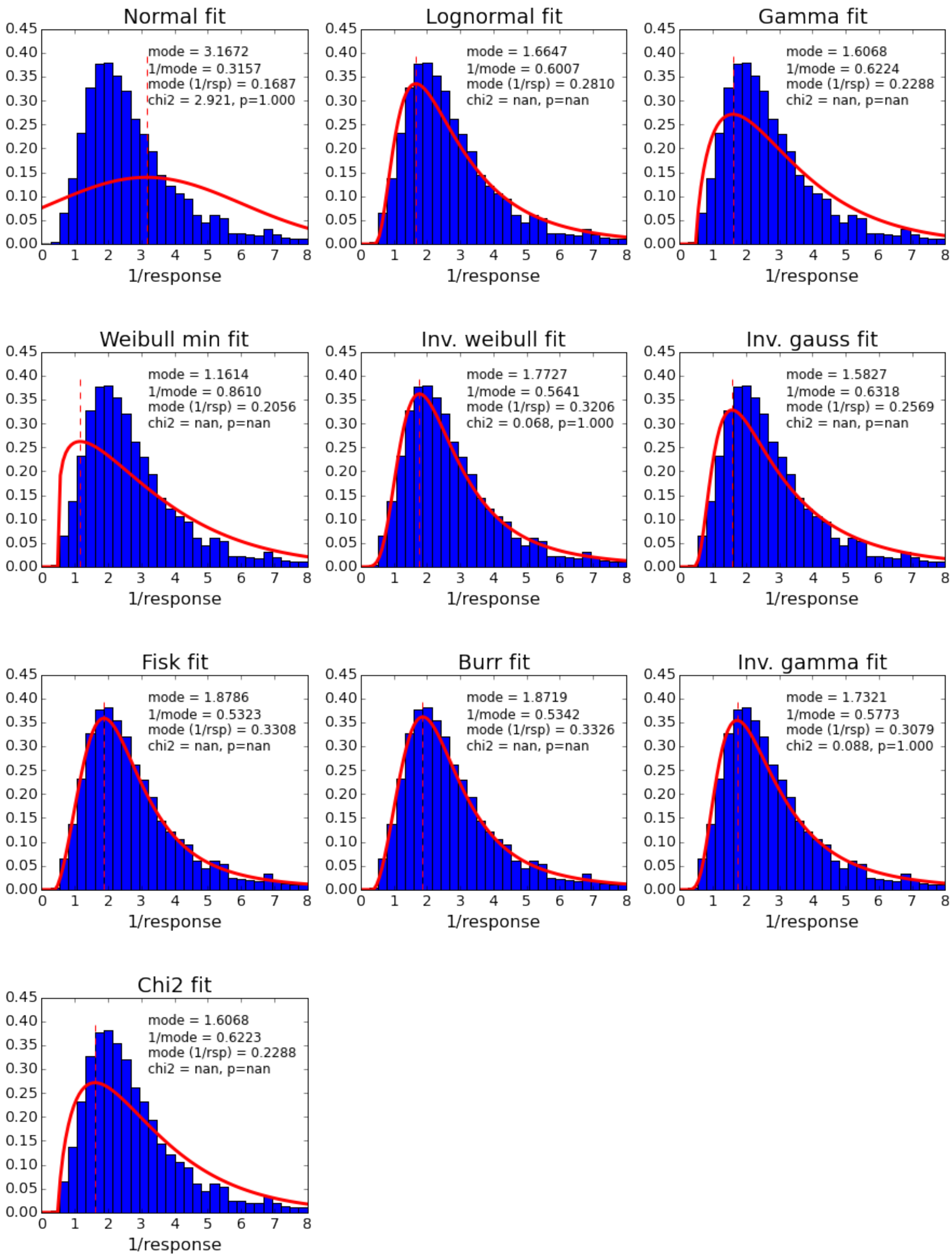



## 1.3 1 / response

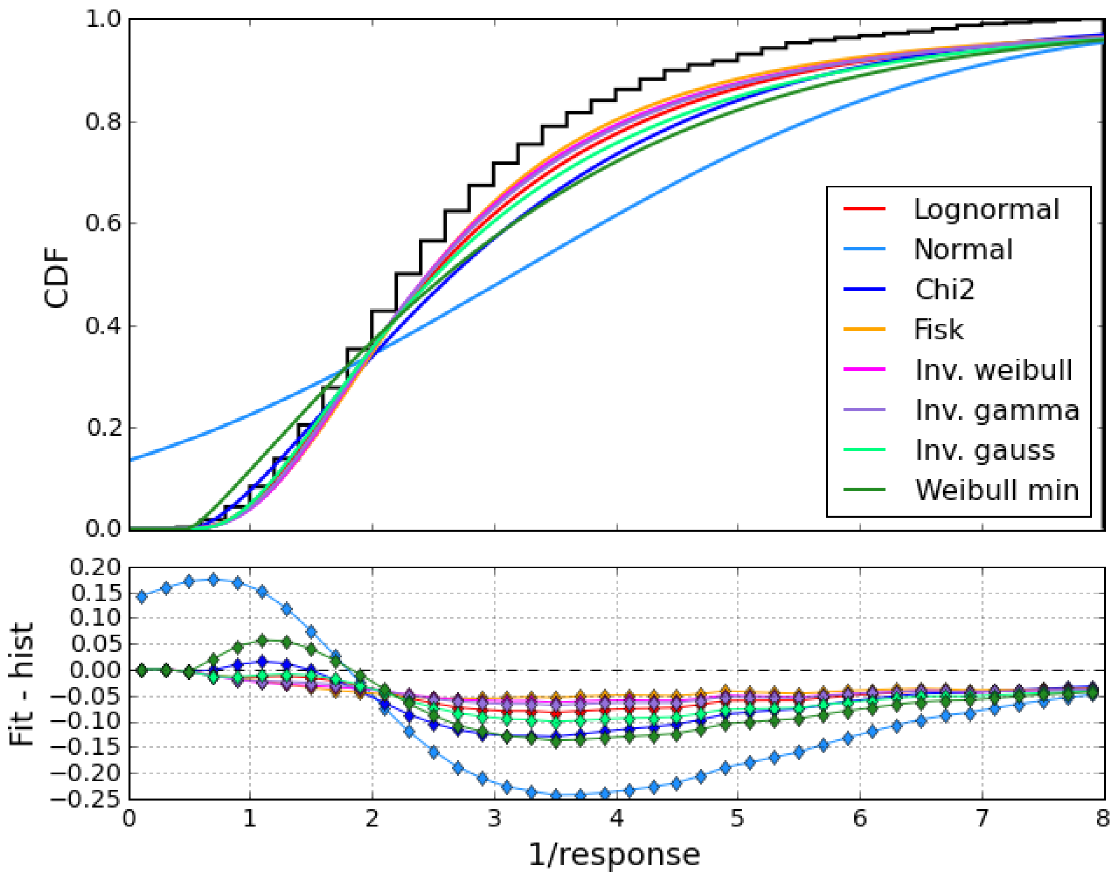
```
In [192]: rspInv_fit_fns = deepcopy(fit_fns_small)
          plot_multiple_fits(rspInv, rspInv_fit_fns, '1/response', [0, 8])
```

```
Doing Normal
```

```
(3.1672435827339647, 2.8594327767627186)
None 3.16724358273 2.85943277676 3.16724352388 0.168733713969 2.9205247368 0.999999998147
Doing Lognormal
(0.69502926870815163, 0.32052388303298784, 2.1790158832281099)
(0.69502926870815163,) 0.320523883033 2.17901588323 1.66473561428 0.281026040356 nan nan
Doing Gamma
(1.7095403085651255, 0.49960218632323633, 1.5604197775989039)
(1.7095403085651255,) 0.499602186323 1.5604197776 1.60678356314 0.228785565293 nan nan
Doing Weibull min
(1.2056090618534028, 0.50000478893152556, 2.8681517184326992)
(1.2056090618534028,) 0.500004788932 2.86815171843 1.16137529676 0.205567020181 nan nan
Doing Inv. weibull
(3.2245199894194867, -1.3771668333998772, 3.425133560238482)
(3.2245199894194867,) -1.3771668334 3.42513356024 1.77271011419 0.320648859338 0.0680928581553 1.0
Doing Inv. gauss
(0.56280024613492385, 0.20831659151827567, 5.2575425967861076)
(0.56280024613492385,) 0.208316591518 5.25754259679 1.58272233503 0.256857799014 nan nan
Doing Fisk
(2.5111087307197542, 0.40563824941211224, 2.0607070723362839)
(2.5111087307197542,) 0.405638249412 2.06070707234 1.87864237554 0.330840782523 nan nan
Doing Burr
(2.4933812341237327, 1.0795428977672512, 0.37423363940865045, 2.0031584574413754)
(2.4933812341237327, 1.0795428977672512) 0.374233639409 2.00315845744 1.87188287306 0.332605356684 nan n
Doing Inv. gamma
(3.7764451226252413, -0.18120128816416164, 9.1387802083809753)
(3.7764451226252413,) -0.181201288164 9.13878020838 1.73210088676 0.30786773858 0.0875259943779 1.0
Doing Chi2
(3.4191275989512917, 0.49959989977113195, 0.78021083736103813)
(3.4191275989512917,) 0.499599899771 0.780210837361 1.60681924489 0.228783244282 nan nan
```




```
In [90]: plot_cdf(rspInv, rspInv_fit_fns, '1/response', [0, 8])
```

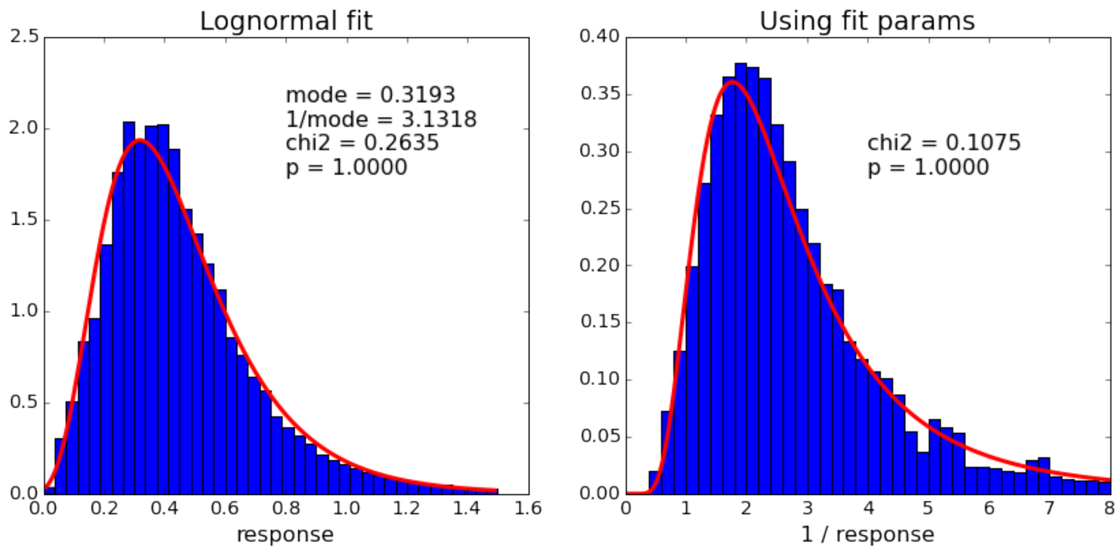



```
In [188]: def apply_fit_to_inverse(data, fit_fn, fn_name):
              """Fit to response, apply function (wiht jacobian) to inverse response."""
              plt.gcf().set_size_inches(14, 6)
              plt.subplot(1, 2, 1)
              ax = plt.gca()

              # cuts for data
              mean = data.mean()
              std = data.std() * 10
              mask = (data < (mean+std)) & (data>(mean-std))

              n, bins, _ = ax.hist(data[mask], bins=40, range=[0, 1.5], normed=True)
              fit_results = fit_fn.fit(data[mask])
              shape = None
              loc = fit_results[-2]
              scale = fit_results[-1]
              if len(fit_results) >=3:
                  shape = fit_results[0:-2]
              print shape, loc, scale
              # plot fitted fn
              x_val = np.arange(0.01, 1.5, 0.01)
              ax.plot(x_val, fit_fn.pdf(x_val, *shape, loc=loc, scale=scale), 'r', linewidth=3)
              ax.set_title('%s fit' % fn_name)
```

```
    ax.set_xlabel('response')

    # get mode
    max_result = minimize(lambda x: -1 * fit_fn.pdf(x, *shape, loc=loc, scale=scale), x0=0.75)
    mode = max_result.x[0]

    # do chi2 test
    bc = get_bin_centers(bins)
    predicted = fit_fn.pdf(bc, *shape, loc=loc, scale=scale)
    ddof = len(shape)+2
    chisq, p = scipy.stats.chisquare(n, f_exp=predicted, ddof=ddof)
    ax.text(0.5, 0.7, 'mode = %.4f\n1/mode = %.4f\nchi2 = %.4f\np = %.4f' % (mode, 1./mode, ch
            transform=ax.transAxes)

    # plot 1/response
    plt.subplot(1, 2, 2)
    ax = plt.gca()
    n, bins, _ = ax.hist(1./data, bins=40, range=[0,8], normed=True)
    x_val = np.arange(0.01, 8, 0.01)
    ax.plot(x_val, np.power((1./x_val), 2) * fit_fn.pdf(1./x_val, *shape, loc=loc, scale=scale
            'r', linewidth=3)

    # do chi2 test
    bc = get_bin_centers(bins)
    predicted = np.power((1./bc), 2) * fit_fn.pdf(1./bc, *shape, loc=loc, scale=scale)
    dof = 3
    chisq, p = scipy.stats.chisquare(n, f_exp=predicted, ddof=dof)

    ax.set_title('Using fit params')
    ax.set_xlabel('1 / response')
    mode = 1.0
    ax.text(0.5, 0.7, 'chi2 = %.4f\np = %.4f' % (chisq, p), transform=ax.transAxes)
```

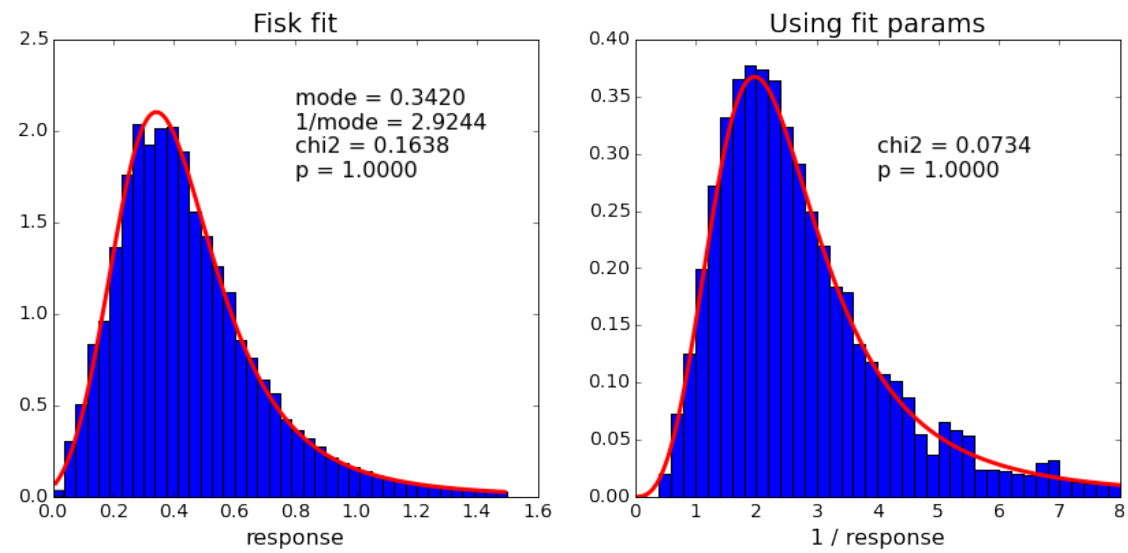
```
In [189]: apply_fit_to_inverse(rsp, scipy.stats.lognorm, 'Lognormal')
```

```
(0.42484031636902841,) -0.124175829665 0.53119838748
```



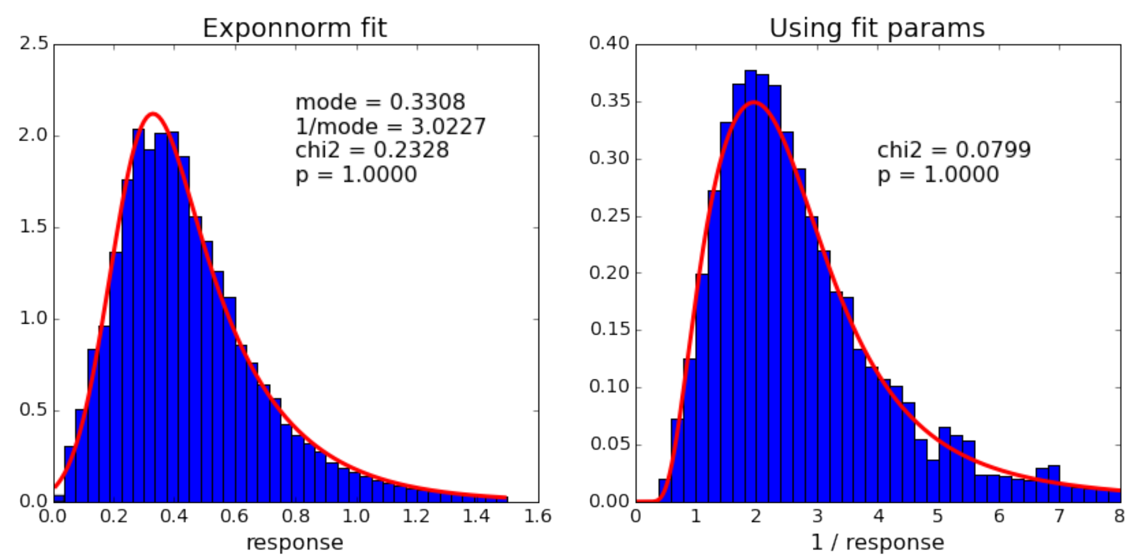
```
In [190]: apply_fit_to_inverse(rsp, scipy.stats.fisk, 'Fisk')
```

```
(3.8624814633240279,) -0.08640309716 0.491336578023
```



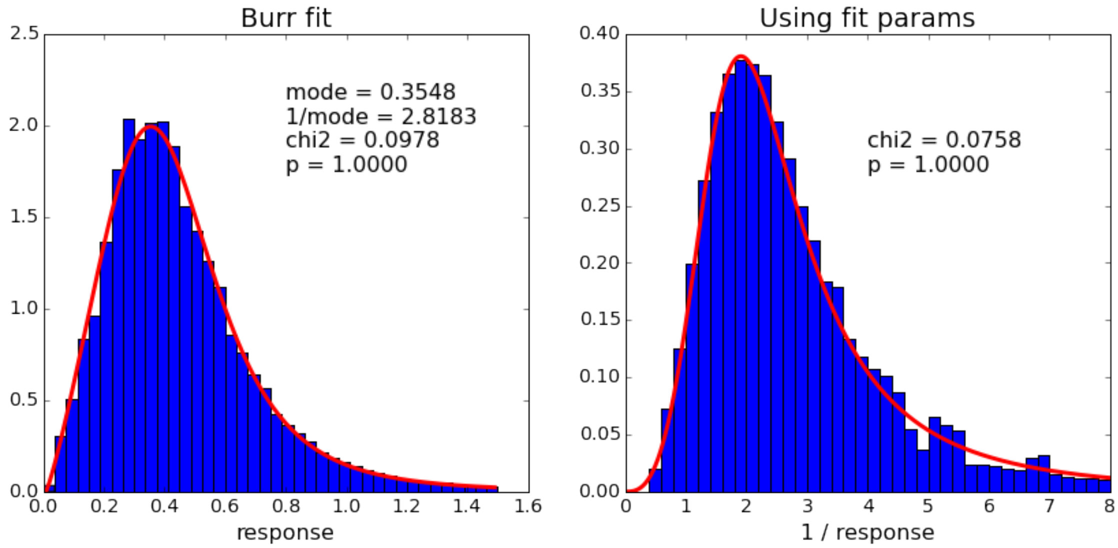

```
In [177]: apply_fit_to_inverse(rsp, scipy.stats.exponnorm, 'Exponnorm')
```

```
(2.2999810196267463,) 0.217451928964 0.104436539053
```




```
In [191]: apply_fit_to_inverse(rsp, scipy.stats.burr, 'Burr')
```

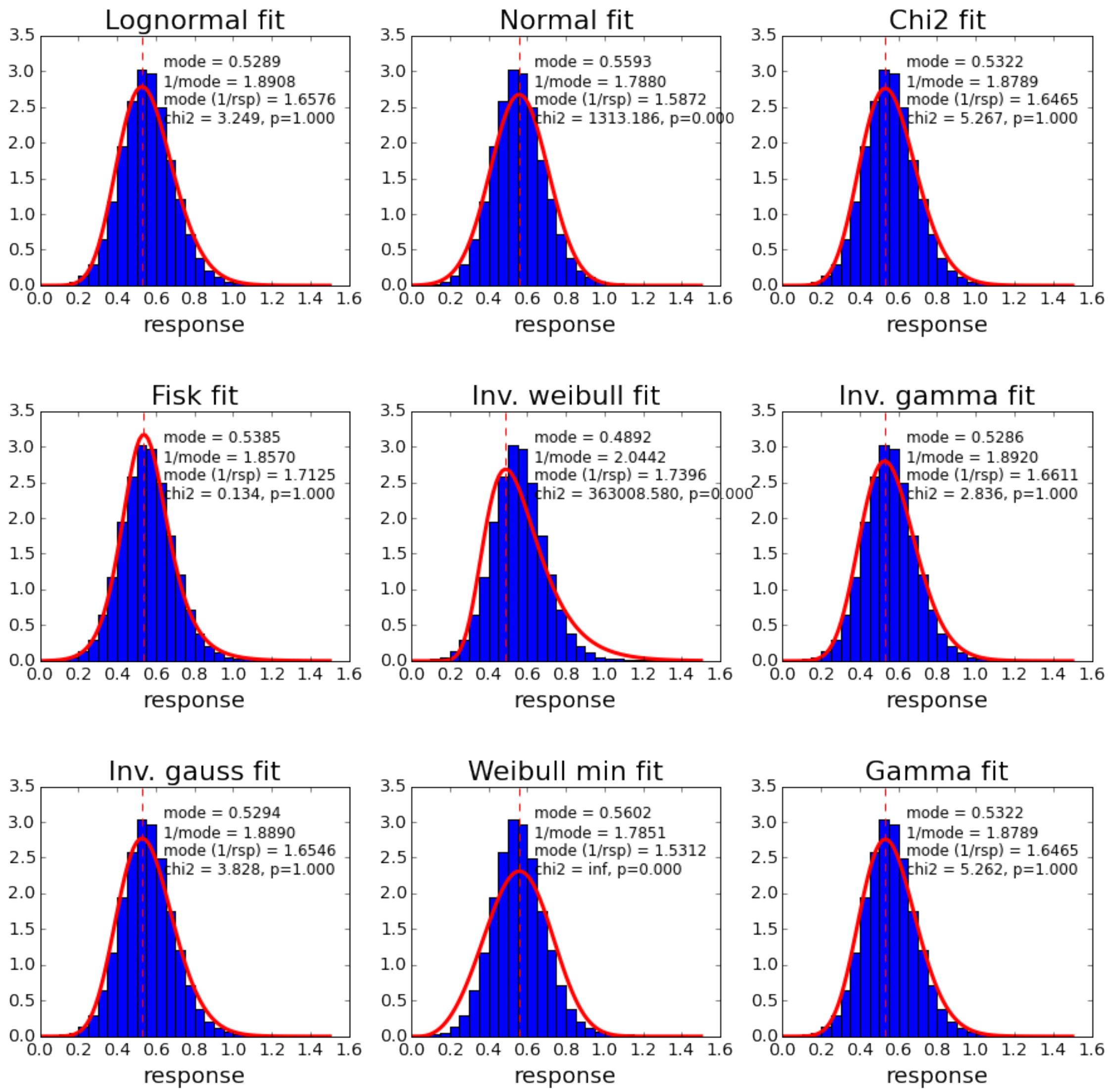
```
(3.7989036602894894, 0.57362014248279913) 0.00912127339789 0.500232499187
```



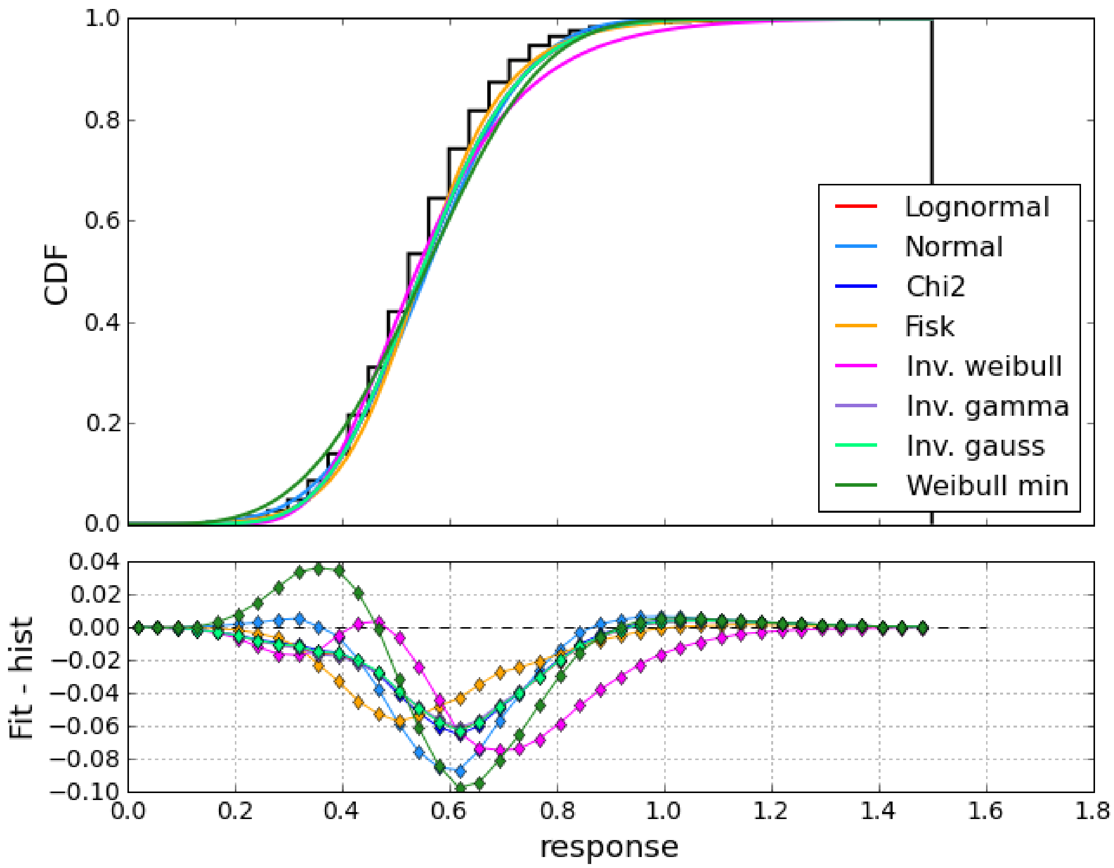

### 1.4 Higher ptRef bin (102 - 106 GeV)

```
In [92]: rspHigh_fit_fns = deepcopy(fit_fns)
         plot_multiple_fits(rspHigh, rspHigh_fit_fns, 'response', [0, 1.5])
```

```
3.24933149492 27
Lognormal : 0.140626234952 -0.480963622578 1.03001070558 0.528877940736 1.65762019654 3.24933149492 0.99
1313.18601662 28
Normal : None 0.559286931472 0.149284479629 0.559286878419 1.58722078147 1313.18601662 6.86568184085e-2
5.26650665297 27
Chi2 : 58.999349671 -0.237433251613 0.0135027829226 0.532216585349 1.64651668568 5.26650665297 0.999995
0.134444607571 27
Fisk : 13.752980815 -0.541426045349 1.09142550297 0.53849940814 1.71249415388 0.134444607571 1.0

/Users/robina/.virtualenvs/ipywidgets/lib/python2.7/site-packages/scipy/stats/_distn_infrastructure.py:1
  return log(self._pdf(x, *args))

363008.579645 27
Inv. weibull : 597849143.997 -82001284.1649 82001284.6541 0.489197778217 1.73962808319 363008.579645 0.
2.83579582188 27
Inv. gamma : 87.8245457636 -0.799475813028 117.961716491 0.528554772027 1.6610818197 2.83579582188 0.99
3.82776876275 27
Inv. gauss : 0.0185603097436 -0.51990259329 58.1289109508 0.529369214926 1.65462095491 3.82776876275 0.

/Users/robina/.virtualenvs/ipywidgets/lib/python2.7/site-packages/ipykernel/__main__.py:65: RuntimeWarnin

inf 27
Weibull min : 3.51354817172 0.0285896700611 0.584779680436 0.560199309935 1.53119507338 inf 0.0
5.26170072418 27
Gamma : 29.5022344198 -0.237508378358 0.0270058046468 0.532217367013 1.64649557408 5.26170072418 0.9999
```

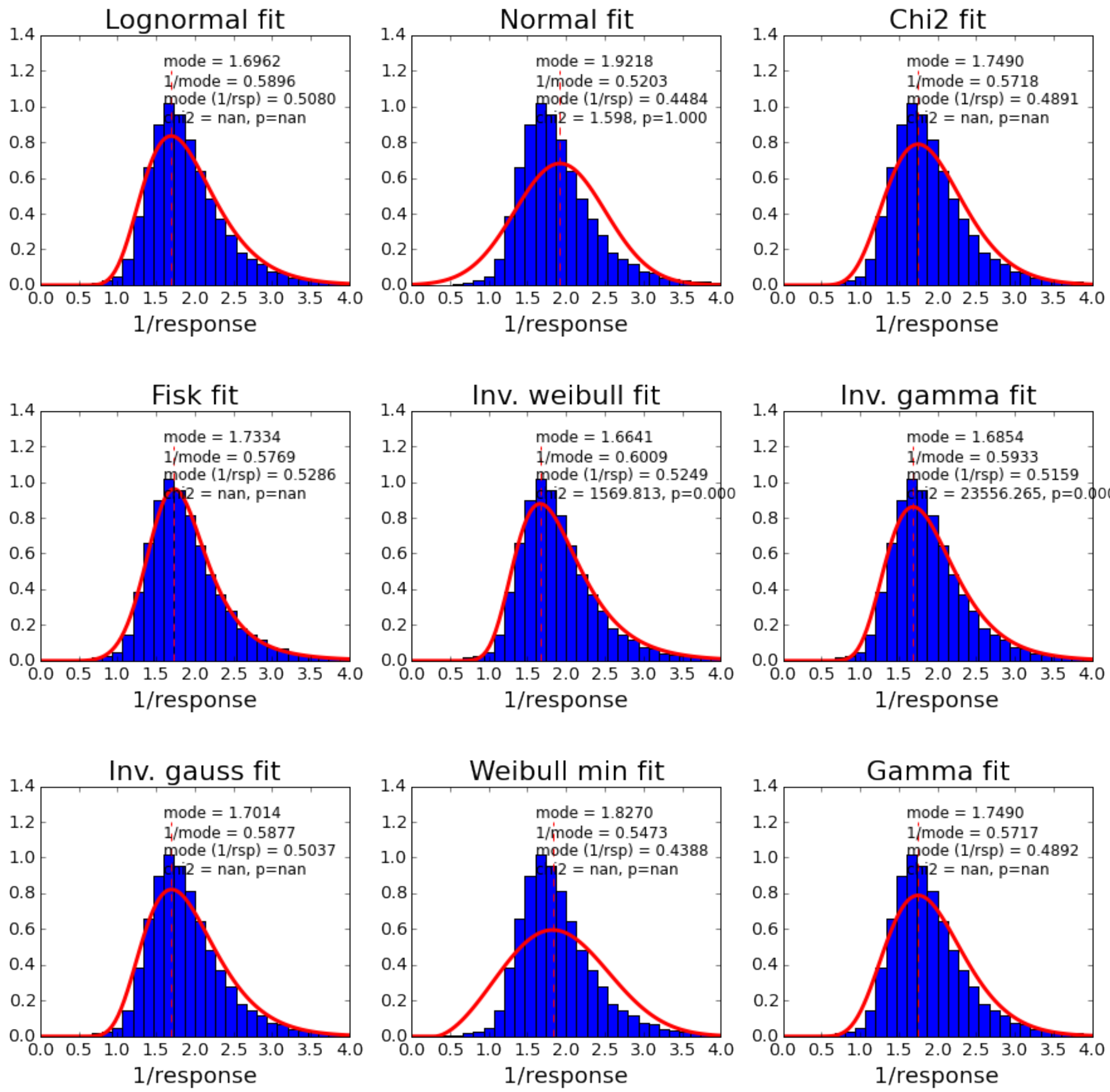



```
In [93]: plot_cdf(rspHigh, rspHigh_fit_fns, 'response', [0, 1.5])
```

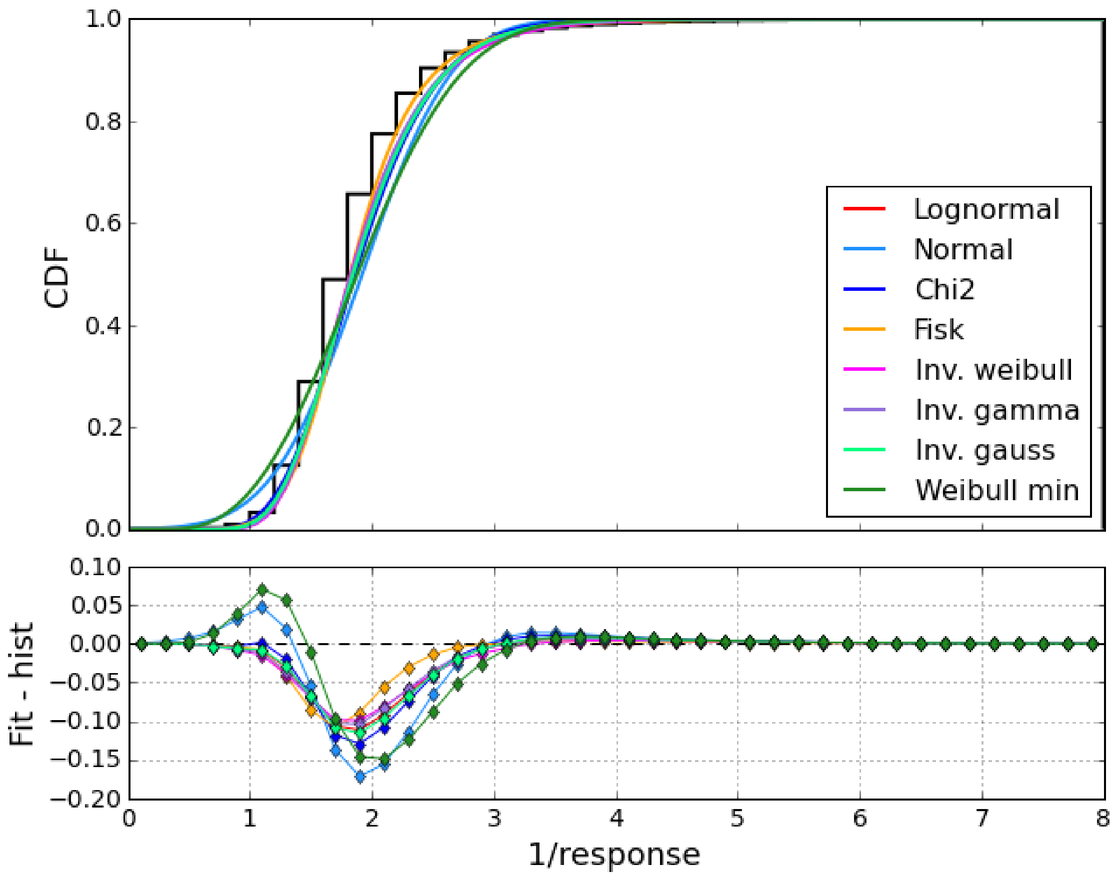



```
In [94]: rspHighInv_fit_fns = deepcopy(fit_fns)
         plot_multiple_fits(rspHighInv, rspHighInv_fit_fns, '1/response', [0, 4])
```

```
/Users/robina/.virtualenvs/ipywidgets/lib/python2.7/site-packages/ipykernel/__main__.py:65: RuntimeWarnin

nan 27
Lognormal : 0.304674920757 0.198004172102 1.64392906104 1.69620101796 0.508009324551 nan nan
1.59796182402 28
Normal : None 1.92182441004 0.586088755972 1.92182418544 0.448448827332 1.59796182402 0.999999999999
nan 27
Chi2 : 18.8545189105 0.294434322436 0.0862995549542 1.74897203024 0.489127673846 nan nan
nan 27
Fisk : 5.74138742917 0.285505732273 1.5393932521 1.73337104536 0.528583859059 nan nan
1569.81339683 27
Inv. weibull : 44.8064669734 -17.0940205659 18.7673690896 1.66410553488 0.524936305728 1569.81339683 0.0
23556.2653776 27
Inv. gamma : 16.8011796389 -0.151153579144 32.6919510291 1.68535111794 0.515882312627 23556.2653776 0.0
nan 27
Inv. gauss : 0.0858855399728 0.0975368219453 21.234993145 1.70143340715 0.503695000403 nan nan
nan 27
Weibull min : 2.70185727828 0.304737528708 1.80626640259 1.82698586259 0.438842803887 nan nan
nan 27
Gamma : 9.44391368153 0.294310055941 0.172280955828 1.74903587096 0.48924047942 nan nan
```

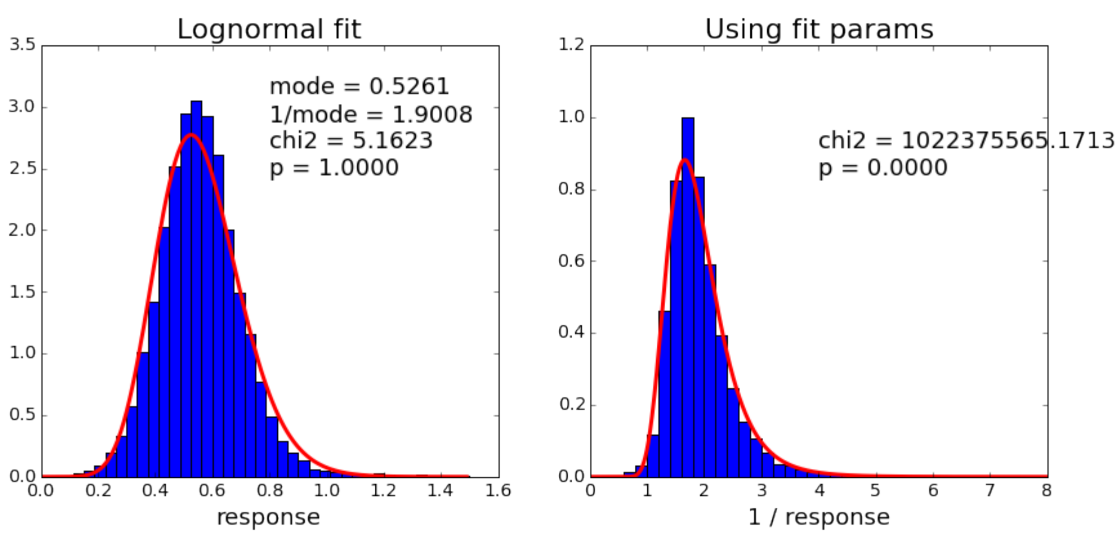



```
In [95]: plot_cdf(rspHighInv, rspHighInv_fit_fns, '1/response', [0, 8])
```



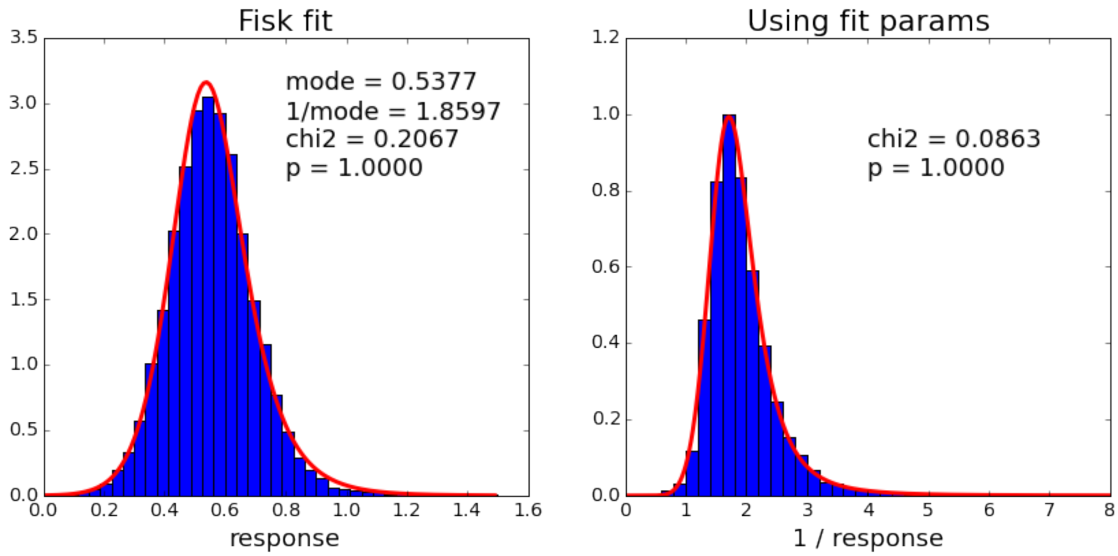

```
In [72]: apply_fit_to_inverse(rspHigh, scipy.stats.lognorm, 'Lognormal')
```

```
0.154298584152 -0.395180479058 0.943463158564
```



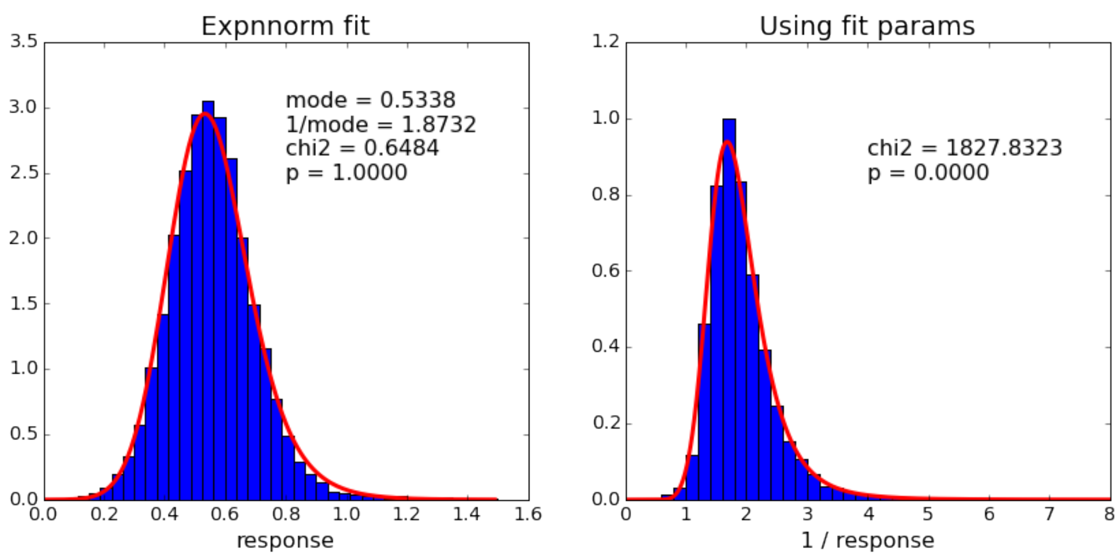
```
In [73]: apply_fit_to_inverse(rspHigh, scipy.stats.fisk, 'Fisk')
```

```
13.0522913071 -0.488558211959 1.03842389756
```



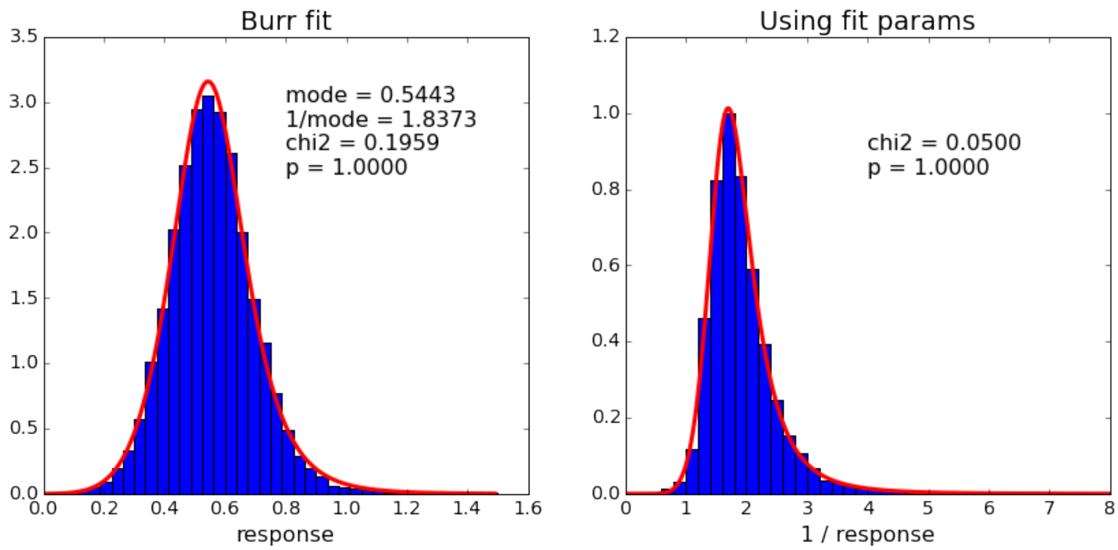

```
In [179]: apply_fit_to_inverse(rspHigh, scipy.stats.exponnorm, 'Expnnorm')
```

```
(0.87557410318578643,) 0.463330557512 0.110058418106
```




```
In [180]: apply_fit_to_inverse(rspHigh, scipy.stats.burr, 'Burr')
```

```
(9.4933700313921499, 0.70746680887276558) -0.113873841711 0.701651954968
```




## 2 Trying my own fitting

In [99]:
```
# For the function
x = np.arange(0.01,10,0.01)


def my_lognorm(x, N, m, theta, sigma):
    x = x[x>theta]
    exp = np.power(np.log((x-theta)/m), 2) / (2 * np.power(sigma, 2))
    result = (N * (x - theta) / ( sigma * np.sqrt(2 * np.pi))) * np.exp(-1. * exp)
    return x, result

def my_gamma(x):
    pass

def my_fisk(x, a, b, c):
    pass
```

In [100]:
```
my_lognorm(x=np.arange(0, 1, 0.2), N=1, m=1, theta=0, sigma=0.5)
```

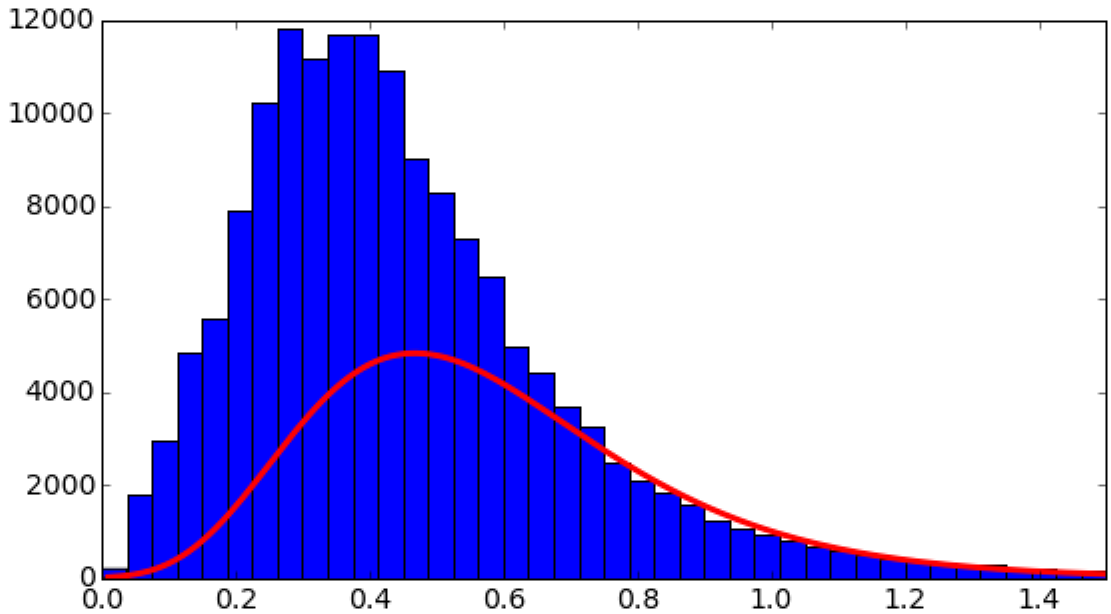
Out[100]:
```
(array([ 0.2,  0.4,  0.6,  0.8]),
 array([ 0.00089758,  0.05953092,  0.28407908,  0.57780375]))
```

In [101]:
```
def plot_hist_fn(hist_data, bins, xlim, x, fn, N, m, theta, sigma):
    plt.hist(hist_data, bins=bins, range=xlim)
    new_x, res = fn(x, N, m, theta, sigma)
    plt.plot(new_x, res, 'r-', linewidth=3)
    plt.xlim(xlim)
```

In [102]:
```
interact(plot_hist_fn, hist_data=fixed(rsp), bins=fixed(40), xlim=fixed([0, 1.5]),
         x=fixed(x),
         fn=fixed(my_lognorm),
         N=widgets.FloatSlider(min=1, max=10000, step=50, value=5851, continuous_update=False
```

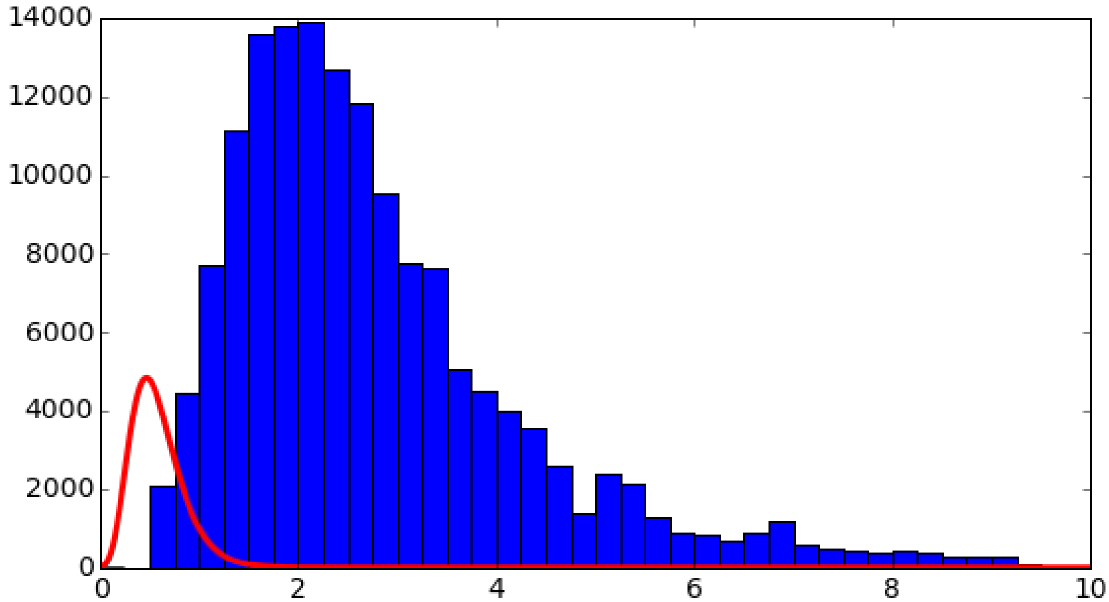
```
                m=widgets.FloatSlider(min=0, max=5, step=0.01, value=0.63, continuous_update=False),
                theta=widgets.FloatSlider(min=-10, max=10, step=0.01, value=-0.23, continuous_update=
                sigma=widgets.FloatSlider(min=0, max=10, step=0.01, value=0.32, continuous_update=Fal
```

Out[102]: <function __main__.plot_hist_fn>



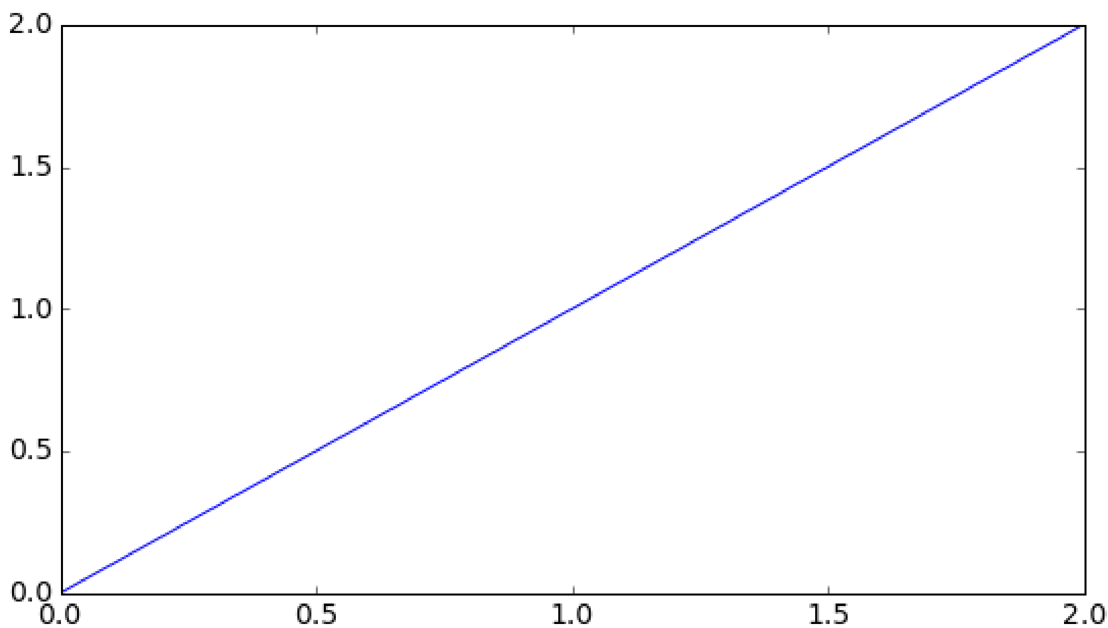

```
In [103]: interact(plot_hist_fn, hist_data=fixed(rspInv), bins=fixed(40), xlim=fixed([0, 10]),
                x=fixed(x),
                fn=fixed(my_lognorm),
                N=widgets.FloatSlider(min=1, max=50000, step=50, value=5851, continuous_update=True)
                m=widgets.FloatSlider(min=0, max=5, step=0.01, value=0.63, continuous_update=True),
                theta=widgets.FloatSlider(min=-10, max=10, step=0.01, value=-0.23, continuous_update=
                sigma=widgets.FloatSlider(min=0, max=10, step=0.01, value=0.32, continuous_update=Tru
```

Out[103]: <function __main__.plot_hist_fn>




```
In [104]: x = np.linspace(0, 2, 100)
          def plot_fisk(a, b, c):
              plt.plot(x, x*a)

          interact(plot_fisk,
                   a=widgets.FloatSlider(min=1, max=5, step=1, value=1, continuous_update=True),
                   b=widgets.FloatSlider(min=1, max=5, step=1, value=1, continuous_update=True),
                   c=widgets.FloatSlider(min=1, max=5, step=1, value=1, continuous_update=True))
```

```
Out[104]: <function __main__.plot_fisk>
```



```
In [ ]:
```