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

Note that we can find a fit for response **or** 1/response - it doesn't matter, since we can transform to either space trivially 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 [196]: 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=40, 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)
```

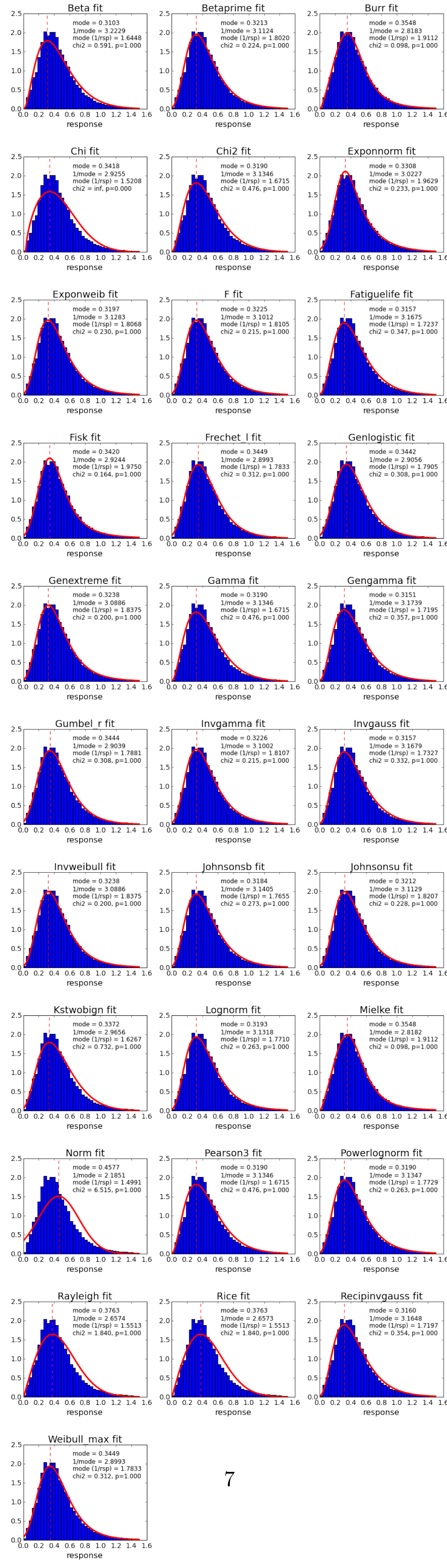
```
                else:
                    frozen_fit = fit_fn_dict['fn'](loc=loc, scale=scale)

                # 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 [199]: 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 0
Doing Betaprime
(16.344245616259741, 10.569644063270601, -0.15111300234983244, 0.35620154983819408)
(16.344245616259741, 10.569644063270601) -0.15111300235 0.356201549838 0.321299454005 1.80203286093 0.2
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 0.476259795836 1.0
Doing Exponnorm
(2.2999810196267463, 0.21745192896381976, 0.10443653905280499)
(2.2999810196267463,) 0.217451928964 0.104436539053 0.330830955551 1.96289747214 0.23276828022 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 0.2
Doing Fatiguelife
(0.42364953467310607, -0.13716766757299764, 0.54586469248275837)
(0.42364953467310607,) -0.137167667573 0.545864692483 0.315711123126 1.72374491566 0.347020050945 1.0
Doing Fisk
(3.8624814633240279, -0.086403097159995418, 0.49133657802261932)
(3.8624814633240279,) -0.08640309716 0.491336578023 0.341950119231 1.97500251507 0.16379384059 1.0
Doing Frechet_l
(7619.3425984471869, 1449.8429883650872, 1449.4980977692076)
(7619.3425984471869,) 1449.84298837 1449.49809777 0.344915458557 1.78327176662 0.311934095635 1.0
Doing Genlogistic
(541.41544304835759, -0.84665747775273892, 0.18919333120968085)
(541.41544304835759,) -0.846657477753 0.18919333121 0.344160603028 1.79050756895 0.307516986026 1.0
Doing Genextreme
(-0.073488485292052513, 0.33681848313975854, 0.18447365110757974)
(-0.073488485292052513,) 0.33681848314 0.184473651108 0.32377076065 1.83752570574 0.200485971932 1.0
Doing Gamma
(3.3319676805752261, -0.0042658391169378288, 0.1386325107522734)
(3.3319676805752261,) -0.00426583911694 0.138632510752 0.319020836187 1.67150639164 0.476256518464 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 0.308116725833 1.0
Doing Invgamma
(9.9648854382754202, -0.28170934898421485, 6.6257912432156356)
(9.9648854382754202,) -0.281709348984 6.62579124322 0.32256428535 1.81068124854 0.215026342036 1.0
Doing Invgauss
(0.18120272930205344, -0.14519334638214776, 3.3269266751309829)
(0.18120272930205344,) -0.145193346382 3.32692667513 0.315670098707 1.73274879573 0.332044166602 1.0
Doing Invweibull
(13.607991237166885, -2.1735203539229007, 2.5103433682933725)
(13.607991237166885,) -2.17352035392 2.51034336829 0.323775532561 1.83750391411 0.200497670702 1.0
Doing Johnsonsb
(10.009559049722476, 2.289140507563352, -0.11800359213495037, 42.131028136357145)
(10.009559049722476, 2.289140507563352) -0.118003592135 42.1310281364 0.318422594467 1.76545017892 0.27
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 0.73160117721 1.0
Doing Lognorm
(0.42484031636902841, -0.12417582966468652, 0.53119838747982184)
(0.42484031636902841,) -0.124175829665 0.53119838748 0.319301307971 1.77096551761 0.263458108913 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 Norm
(0.45765475257061072, 0.26428265684062685)
None 0.457654752571 0.264282656841 0.457654745345 1.49910247805 6.51539046798 0.999999994874
```

```
Doing Pearson3
(1.0956662411628391, 0.45765071897792242, 0.25305337759380087)
(1.0956662411628391,) 0.457650718978 0.253053377594 0.319019838649 1.67151707009 0.476238413733 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 1.83950799224 1.0
Doing Rice
(0.00094108302785169784, 0.0067459713859989844, 0.3695699844982272)
(0.00094108302785169784,) 0.006745971386 0.369569984498 0.376316014981 1.55129731958 1.83950265135 1.0
Doing Recipinvgauss
(0.19684433222182118, -0.12619423832387205, 0.096025475590688011)
(0.19684433222182118,) -0.126194238324 0.0960254755907 0.315974518158 1.71968473538 0.353605186613 1.0
Doing Weibull_max
(7619.3425984471869, 1449.8429883650872, 1449.4980977692076)
(7619.3425984471869,) 1449.84298837 1449.49809777 0.344915458557 1.78327176662 0.311934095635 1.0
```


Beta fit
mode = 0.3103
1/mode = 3.2229
mode (1/rsp) = 1.6448
chi2 = 0.591, p=1.000
Betaprime fit
mode = 0.3213
1/mode = 3.1124
mode (1/rsp) = 1.8020
chi2 = 0.224, p=1.000
Burr fit
mode = 0.3548
1/mode = 2.8183
mode (1/rsp) = 1.9112
chi2 = 0.098, 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 = 0.476, p=1.000
Exponnorm fit
mode = 0.3308
1/mode = 3.0227
mode (1/rsp) = 1.9629
chi2 = 0.233, p=1.000
Exponweib fit
mode = 0.3197
1/mode = 3.1283
mode (1/rsp) = 1.8068
chi2 = 0.230, p=1.000
F fit
mode = 0.3225
1/mode = 3.1012
mode (1/rsp) = 1.8105
chi2 = 0.215, p=1.000
Fatiguelife fit
mode = 0.3157
1/mode = 3.1675
mode (1/rsp) = 1.7237
chi2 = 0.347, p=1.000
Fisk fit
mode = 0.3420
1/mode = 2.9244
mode (1/rsp) = 1.9750
chi2 = 0.164, p=1.000
Frechet_l fit
mode = 0.3449
1/mode = 2.8993
mode (1/rsp) = 1.7833
chi2 = 0.312, p=1.000
Genlogistic fit
mode = 0.3442
1/mode = 2.9056
mode (1/rsp) = 1.7905
chi2 = 0.308, p=1.000
Genextreme fit
mode = 0.3238
1/mode = 3.0886
mode (1/rsp) = 1.8375
chi2 = 0.200, p=1.000
Gamma fit
mode = 0.3190
1/mode = 3.1346
mode (1/rsp) = 1.6715
chi2 = 0.476, p=1.000
Gengamma fit
mode = 0.3151
1/mode = 3.1739
mode (1/rsp) = 1.7195
chi2 = 0.357, p=1.000
Gumbel_r fit
mode = 0.3444
1/mode = 2.9039
mode (1/rsp) = 1.7881
chi2 = 0.308, p=1.000
Invgamma fit
mode = 0.3226
1/mode = 3.1002
mode (1/rsp) = 1.8107
chi2 = 0.215, p=1.000
Invgauss fit
mode = 0.3157
1/mode = 3.1679
mode (1/rsp) = 1.7327
chi2 = 0.332, p=1.000
Invweibull fit
mode = 0.3238
1/mode = 3.0886
mode (1/rsp) = 1.8375
chi2 = 0.200, p=1.000
Johnsonsb fit
mode = 0.3184
1/mode = 3.1405
mode (1/rsp) = 1.7655
chi2 = 0.273, p=1.000
Johnsonsu fit
mode = 0.3212
1/mode = 3.1129
mode (1/rsp) = 1.8207
chi2 = 0.228, p=1.000
Kstwobign fit
mode = 0.3372
1/mode = 2.9656
mode (1/rsp) = 1.6267
chi2 = 0.732, p=1.000
Lognorm fit
mode = 0.3193
1/mode = 3.1318
mode (1/rsp) = 1.7710
chi2 = 0.263, p=1.000
Mielke fit
mode = 0.3548
1/mode = 2.8182
mode (1/rsp) = 1.9112
chi2 = 0.098, p=1.000
Norm fit
mode = 0.4577
1/mode = 2.1851
mode (1/rsp) = 1.4991
chi2 = 6.515, p=1.000
Pearson3 fit
mode = 0.3190
1/mode = 3.1346
mode (1/rsp) = 1.6715
chi2 = 0.476, p=1.000
Powerlognorm fit
mode = 0.3190
1/mode = 3.1347
mode (1/rsp) = 1.7729
chi2 = 0.263, p=1.000
Rayleigh fit
mode = 0.3763
1/mode = 2.6574
mode (1/rsp) = 1.5513
chi2 = 1.840, p=1.000
Rice fit
mode = 0.3763
1/mode = 2.6573
mode (1/rsp) = 1.5513
chi2 = 1.840, p=1.000
Recipinvgauss fit
mode = 0.3160
1/mode = 3.1648
mode (1/rsp) = 1.7197
chi2 = 0.354, p=1.000
Weibull_max fit
mode = 0.3449
1/mode = 2.8993
mode (1/rsp) = 1.7833
chi2 = 0.312, 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 [200]:
```
print_ordered_fit_fn(rsp_fit_fns)
```

```
Burr 0.0978116451796 1.0
Mielke 0.0978159600107 1.0
Fisk 0.16379384059 1.0
Genextreme 0.200485971932 1.0
Invweibull 0.200497670702 1.0
F 0.215024069555 1.0
Invgamma 0.215026342036 1.0
Betaprime 0.22420666793 1.0
Johnsonsu 0.228140502355 1.0
Exponweib 0.230204187685 1.0
Exponnorm 0.23276828022 1.0
Powerlognorm 0.263131296688 1.0
Lognorm 0.263458108913 1.0
Johnsonsb 0.272677858578 1.0
Genlogistic 0.307516986026 1.0
Gumbel_r 0.308116725833 1.0
Frechet_l 0.311934095635 1.0
Weibull_max 0.311934095635 1.0
Invgauss 0.332044166602 1.0
Fatiguelife 0.347020050945 1.0
Recipinvgauss 0.353605186613 1.0
Gengamma 0.356545059663 1.0
Pearson3 0.476238413733 1.0
Gamma 0.476256518464 1.0
Chi2 0.476259795836 1.0
Beta 0.591062013024 1.0
Kstwobign 0.73160117721 1.0
Rice 1.83950265135 1.0
Rayleigh 1.83950799224 1.0
Norm 6.51539046798 0.999999994874
Chi inf 0.0
```

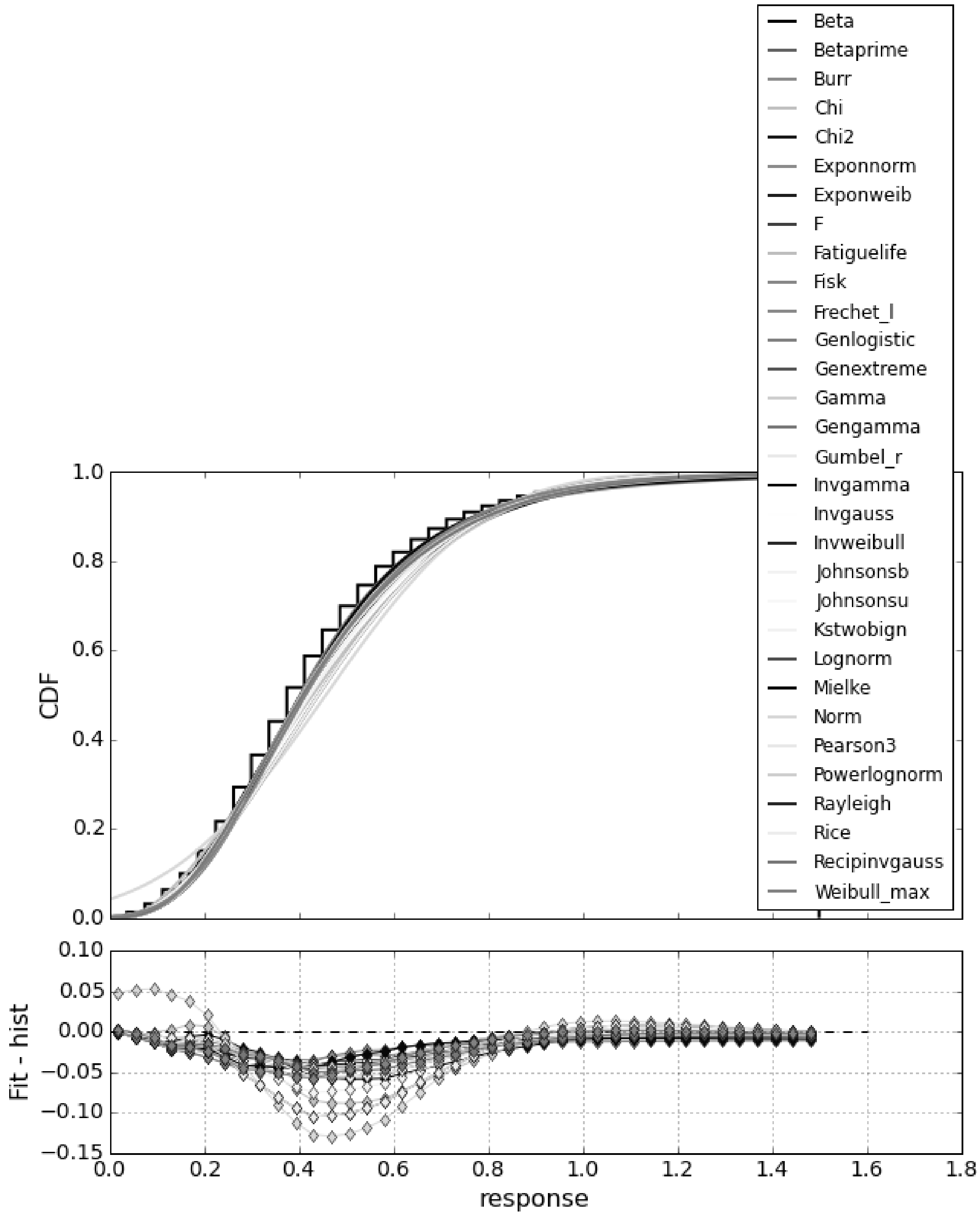
In [86]:
```
def calc_hist_fn_diff(n, bins, fn):
    centers = get_bin_centers(bins)
    fn_vals = np.array([fn(x) for x in centers])
    return fn_vals - n
```

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

```
[  2.32355432e-04   3.82921784e-01   5.47812497e-01   7.52306979e-01
   9.31384438e-02   5.47042104e-01   1.15064286e-01   2.52607285e-01
   7.43743009e-01   5.30664325e-01   5.38595516e-01   5.00225991e-01
   3.32799209e-01   8.07367742e-01   4.67412246e-01   9.14904634e-01
   3.12690332e-02   9.94485934e-01   1.57607027e-01   9.48517456e-01
   9.68778440e-01   9.49673913e-01   3.03957568e-01   3.16458225e-02
   8.41726710e-01   9.09595506e-01   8.09398313e-01   1.47071983e-01
   9.29326989e-01   4.63523394e-01   5.04208601e-01]
```



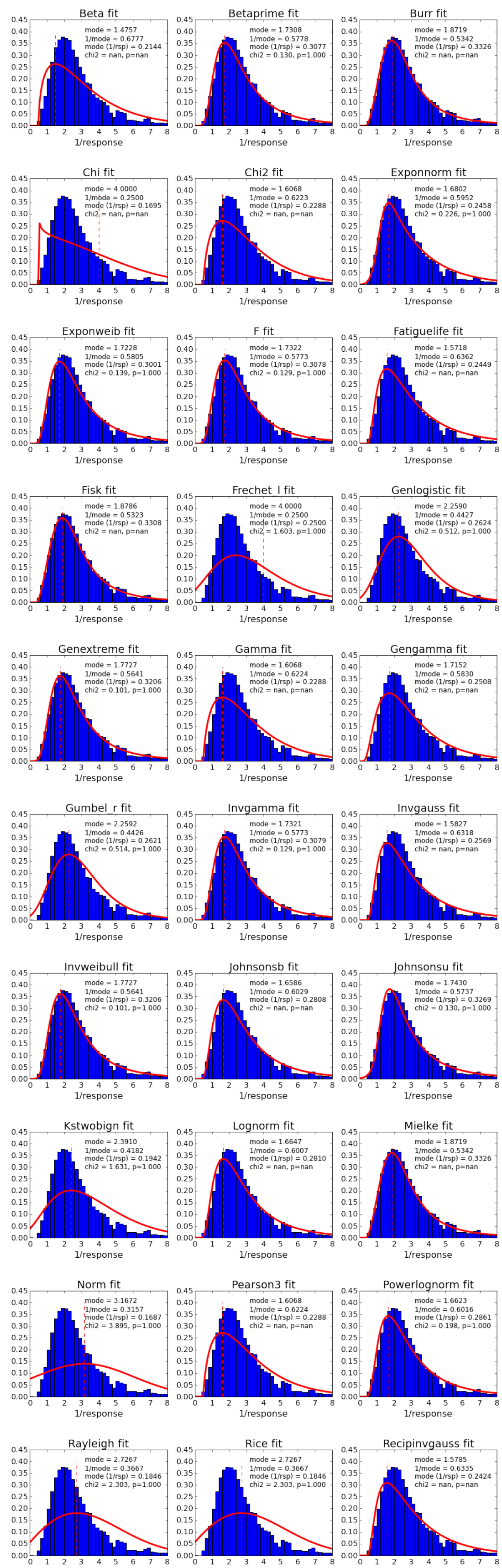
1.3 1 / response

```
In [202]: rspInv_fit_fns = deepcopy(fit_fns)
          plot_multiple_fits(rspInv, rspInv_fit_fns, '1/response', [0, 8])
```

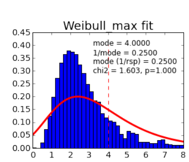
```
Doing Beta
(1.5464057176090531, 57530.425713092292, 0.49989595424279998, 102735.83814050592)
```

```
(1.5464057176090531, 57530.425713092292) 0.499895954243 102735.838141 1.47565569954 0.214394232421 nan
Doing Betaprime
(34.081921835496459, 3.7895868109746691, -0.06153832774798168, 0.25948925010943957)
(34.081921835496459, 3.7895868109746691) -0.061538327748 0.259489250109 1.7307681233 0.307665471785 0.1
Doing Burr
(2.4933812341237327, 1.0795428977672512, 0.37423363940865045, 2.0031584574413754)
(2.4933812341237327, 1.0795428977672512) 0.374233639409 2.00315845744 1.87188287306 0.332605356684 nan
Doing Chi
(0.91656816143329745, 0.50001167527261747, 4.0849170647648254)
(0.91656816143329745,) 0.500011675273 4.08491706476 4.0 0.169511748049 nan nan
Doing Chi2
(3.4191275989512917, 0.49959989977113195, 0.78021083736103813)
(3.4191275989512917,) 0.499599899771 0.780210837361 1.60681924489 0.228783244282 nan nan
Doing Exponnorm
(5.6704556095259742, 1.1332564626787449, 0.35869932710778729)
(5.6704556095259742,) 1.13325646268 0.358699327108 1.68022714542 0.245822412914 0.225829871149 1.0
Doing Exponweib
(126.11313701067678, 0.38133902244432277, -0.018329640065858799, 0.033057266957980697)
(126.11313701067678, 0.38133902244432277) -0.0183296400659 0.033057266958 1.72276575024 0.300146112219
Doing F
(2145.4658844512473, 7.5520866384060419, -0.17699295865634224, 2.4169917647874035)
(2145.4658844512473, 7.5520866384060419) -0.176992958656 2.41699176479 1.73216273721 0.307831379392 0.1
Doing Fatiguelife
(0.7200140142957927, 0.23932924755463725, 2.3313804102168341)
(0.7200140142957927,) 0.239329247555 2.33138041022 1.57176816259 0.244901491631 nan nan
Doing Fisk
(2.5111087307197542, 0.40563824941211224, 2.0607070723362839)
(2.5111087307197542,) 0.405638249412 2.06070707234 1.87864237554 0.330840782523 nan nan
Doing Frechet_l
(4148497098.7707577, 7642822879.1176119, 7642822876.7269974)
(4148497098.7707577,) 7642822879.12 7642822876.73 4.0 0.25 1.60328704889 1.0
Doing Genlogistic
(988.8694589872141, -6.8303498598257839, 1.3179635970140318)
(988.8694589872141,) -6.83034985983 1.31796359701 2.25900847121 0.262411687897 0.511868989305 1.0
Doing Genextreme
(-0.31013636268208866, 2.0479606878034637, 1.0622193223262006)
(-0.31013636268208866,) 2.0479606878 1.06221932233 1.77269394138 0.32064850213 0.101221990303 1.0
Doing Gamma
(1.7095403085651255, 0.49960218632323633, 1.5604197775989039)
(1.7095403085651255,) 0.499602186323 1.5604197776 1.60678356314 0.228785565293 nan nan
Doing Gengamma
(11.636913220859668, 0.42822665179661312, 0.1904877115091671, 0.0083443205319264739)
(11.636913220859668, 0.42822665179661312) 0.190487711509 0.00834432053193 1.71522738237 0.250752727084
Doing Gumbel_r
(2.2592878933499136, 1.3202443816025202)
None 2.25928789335 1.3202443816 2.2592362826 0.262136630226 0.5143965236 1.0
Doing Invgamma
(3.7764451226252413, -0.18120128816416164, 9.1387802083809753)
(3.7764451226252413,) -0.181201288164 9.13878020838 1.73210088676 0.30786773858 0.129062452187 1.0
Doing Invgauss
(0.56280024613492385, 0.20831659151827567, 5.2575425967861076)
(0.56280024613492385,) 0.208316591518 5.25754259679 1.58272233503 0.256857799014 nan nan
Doing Invweibull
(3.2245199894194867, -1.3771668333998772, 3.425133560238482)
```

```
(3.2245199894194867,) -1.3771668334 3.42513356024 1.77271011419 0.320648859338 0.101214786037 1.0
Doing Johnsonsb
(9.1804451030466687, 1.4299667985116207, 0.32513901898626796, 1335.2727309916245)
(9.1804451030466687, 1.4299667985116207) 0.325139018986 1335.27273099 1.65860780904 0.280796590451 nan n
Doing Johnsonsu
(-1.7285924776935817, 1.1359029115494805, 1.0339793615019843, 0.64587258329555308)
(-1.7285924776935817, 1.1359029115494805) 1.0339793615 0.645872583296 1.74299565554 0.326889712418 0.12
Doing Kstwobign
(-3.7792166553948596, 8.3895570335147944)
None -3.77921665539 8.38955703351 2.39095173201 0.19424427591 1.63076789514 1.0
Doing Lognorm
(0.69502926870815163, 0.32052388303298784, 2.1790158832281099)
(0.69502926870815163,) 0.320523883033 2.17901588323 1.66473561428 0.281026040356 nan nan
Doing Mielke
(2.6916760490078535, 2.4933695161203611, 0.37424657095067371, 2.0031540961451983)
(2.6916760490078535, 2.4933695161203611) 0.374246570951 2.00315409615 1.8718797454 0.332605499905 nan n
Doing Norm
(3.1672435827339647, 2.8594327767627186)
None 3.16724358273 2.85943277676 3.16724352388 0.168733713969 3.89456185708 0.999999999999
Doing Pearson3
(1.5296430149174833, 3.1672142387059505, 2.0402487857892191)
(1.5296430149174833,) 3.16721423871 2.04024878579 1.60678872533 0.228784536078 nan nan
Doing Powerlognorm
(0.2537167763962575, 0.38722405298954859, 0.090207912879439703, 1.3349256283175555)
(0.2537167763962575, 0.38722405298954859) 0.0902079128794 1.33492562832 1.66228599296 0.286067986605 0.
Doing Rayleigh
(-0.64038189807084822, 3.3670850445472782)
None -0.640381898071 3.36708504455 2.72669876939 0.184602860495 2.30319178019 1.0
Doing Rice
(0.00078242047621178922, -0.64037165725836798, 3.3670665579123109)
(0.00078242047621178922,) -0.640371657258 3.36706655791 2.72669117891 0.184603699371 2.30316063672 1.0
Doing Recipinvgauss
(0.63874005153676361, 0.30104355596177601, 1.1171544611906499)
(0.63874005153676361,) 0.301043555962 1.11715446119 1.57849464524 0.242354518004 nan nan
Doing Weibull_max
(4148497098.7707577, 7642822879.1176119, 7642822876.7269974)
(4148497098.7707577,) 7642822879.12 7642822876.73 4.0 0.25 1.60328704889 1.0
```



Beta fit
mode = 1.4757
1/mode = 0.6777
mode (1/rsp) = 0.2144
chi2 = nan, p=nan
Betaprime fit
mode = 1.7308
1/mode = 0.5778
mode (1/rsp) = 0.3077
chi2 = 0.130, p=1.000
Burr fit
mode = 1.8719
1/mode = 0.5342
mode (1/rsp) = 0.3326
chi2 = nan, p=nan
Chi fit
mode = 4.0000
1/mode = 0.2500
mode (1/rsp) = 0.1695
chi2 = nan, p=nan
Chi2 fit
mode = 1.6068
1/mode = 0.6223
mode (1/rsp) = 0.2288
chi2 = nan, p=nan
Exponnorm fit
mode = 1.6802
1/mode = 0.5952
mode (1/rsp) = 0.2458
chi2 = 0.226, p=1.000
Exponweib fit
mode = 1.7228
1/mode = 0.5805
mode (1/rsp) = 0.3001
chi2 = 0.139, p=1.000
F fit
mode = 1.7322
1/mode = 0.5773
mode (1/rsp) = 0.3078
chi2 = 0.129, p=1.000
Fatiguelife fit
mode = 1.5718
1/mode = 0.6362
mode (1/rsp) = 0.2449
chi2 = nan, p=nan
Fisk fit
mode = 1.8786
1/mode = 0.5323
mode (1/rsp) = 0.3308
chi2 = nan, p=nan
Frechet_l fit
mode = 4.0000
1/mode = 0.2500
mode (1/rsp) = 0.2500
chi2 = 1.603, p=1.000
Genlogistic fit
mode = 2.2590
1/mode = 0.4427
mode (1/rsp) = 0.2624
chi2 = 0.512, p=1.000
Genextreme fit
mode = 1.7727
1/mode = 0.5641
mode (1/rsp) = 0.3206
chi2 = 0.101, p=1.000
Gamma fit
mode = 1.6068
1/mode = 0.6224
mode (1/rsp) = 0.2288
chi2 = nan, p=nan
Gengamma fit
mode = 1.7152
1/mode = 0.5830
mode (1/rsp) = 0.2508
chi2 = nan, p=nan
Gumbel_r fit
mode = 2.2592
1/mode = 0.4426
mode (1/rsp) = 0.2621
chi2 = 0.514, p=1.000
Invgamma fit
mode = 1.7321
1/mode = 0.5773
mode (1/rsp) = 0.3079
chi2 = 0.129, p=1.000
Invgauss fit
mode = 1.5827
1/mode = 0.6318
mode (1/rsp) = 0.2569
chi2 = nan, p=nan
Invweibull fit
mode = 1.7727
1/mode = 0.5641
mode (1/rsp) = 0.3206
chi2 = 0.101, p=1.000
Johnsonsb fit
mode = 1.6586
1/mode = 0.6029
mode (1/rsp) = 0.2808
chi2 = nan, p=nan
Johnsonsu fit
mode = 1.7430
1/mode = 0.5737
mode (1/rsp) = 0.3269
chi2 = 0.130, p=1.000
Kstwobign fit
mode = 2.3910
1/mode = 0.4182
mode (1/rsp) = 0.1942
chi2 = 1.631, p=1.000
Lognorm fit
mode = 1.6647
1/mode = 0.6007
mode (1/rsp) = 0.2810
chi2 = nan, p=nan
Mielke fit
mode = 1.8719
1/mode = 0.5342
mode (1/rsp) = 0.3326
chi2 = nan, p=nan
Norm fit
mode = 3.1672
1/mode = 0.3157
mode (1/rsp) = 0.1687
chi2 = 3.895, p=1.000
Pearson3 fit
mode = 1.6068
1/mode = 0.6224
mode (1/rsp) = 0.2288
chi2 = nan, p=nan
Powerlognorm fit
mode = 1.6623
1/mode = 0.6016
mode (1/rsp) = 0.2861
chi2 = 0.198, p=1.000
Rayleigh fit
mode = 2.7267
1/mode = 0.3667
mode (1/rsp) = 0.1846
chi2 = 2.303, p=1.000
Rice fit
mode = 2.7267
1/mode = 0.3667
mode (1/rsp) = 0.1846
chi2 = 2.303, p=1.000
Recipinvgauss fit
mode = 1.5785
1/mode = 0.6335
mode (1/rsp) = 0.2424
chi2 = nan, p=nan
1/response



Weibull_max fit
mode = 4.0000
1/mode = 0.2500
mode (1/rsp) = 0.2500
chi2 = 1.603, p=1.000
1/response

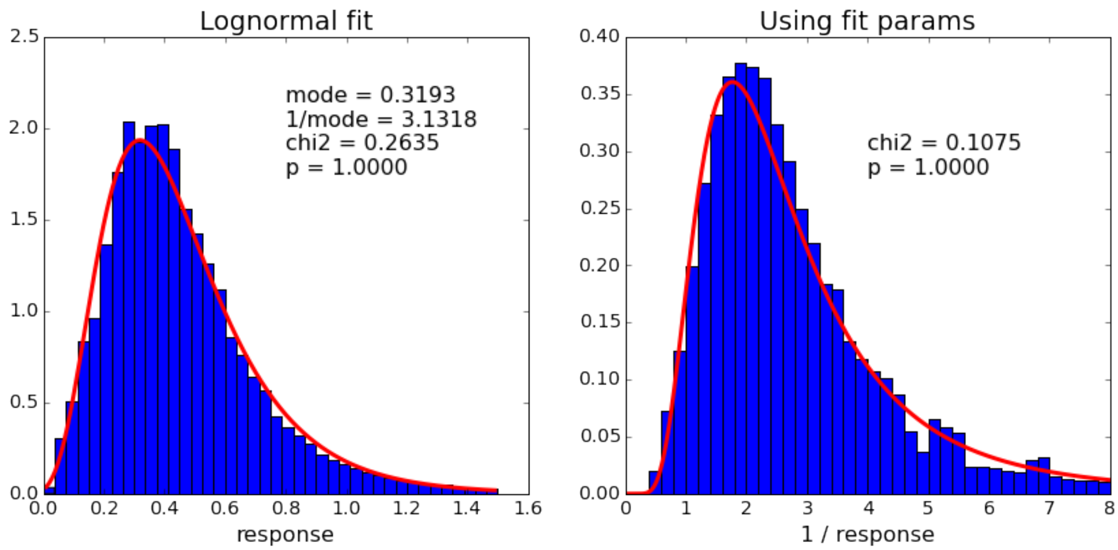
In [188]:
```
def apply_fit_to_inverse(data, fit_fn, fn_name):
    """Fit to response, apply function (wiht jacobian) to inverse response."""
    plt.gcf().set_size_inches(14, 6)
    plt.subplot(1, 2, 1)
    ax = plt.gca()

    # 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, c
            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=scal
            '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)
```

We can check how well the distribution models response, by plotting it on top of 1/response (with necessary Jacobian transform) and calcualting chi2.

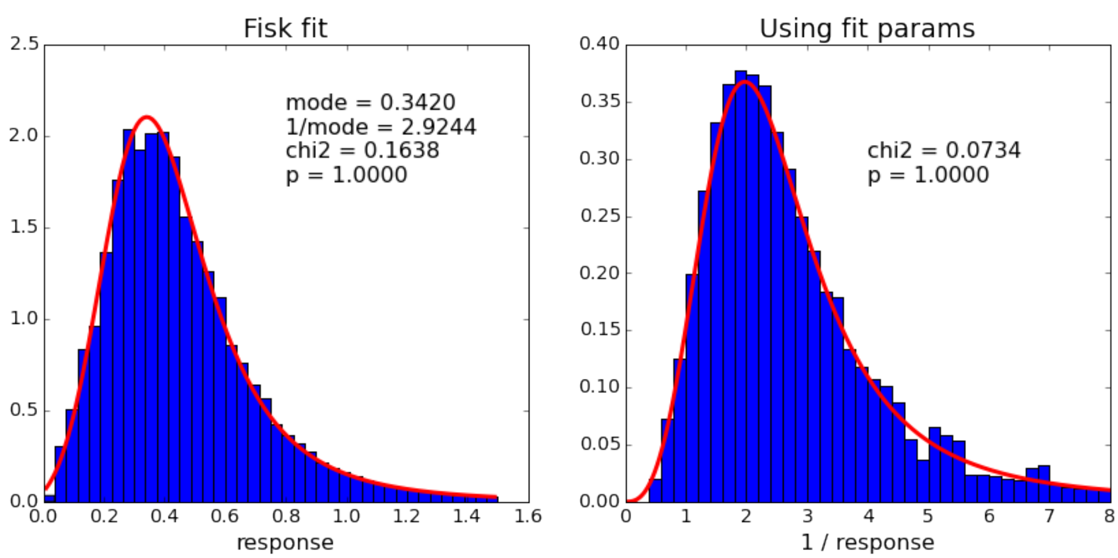
```
In [189]: apply_fit_to_inverse(rsp, scipy.stats.lognorm, 'Lognormal')
```

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



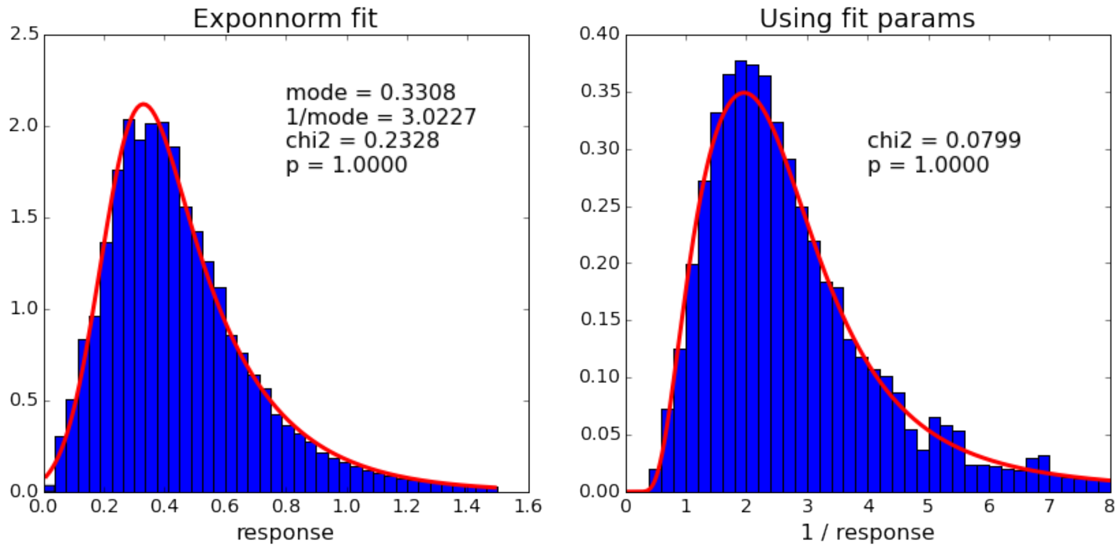

```
In [190]: apply_fit_to_inverse(rsp, scipy.stats.fisk, 'Fisk')
```

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



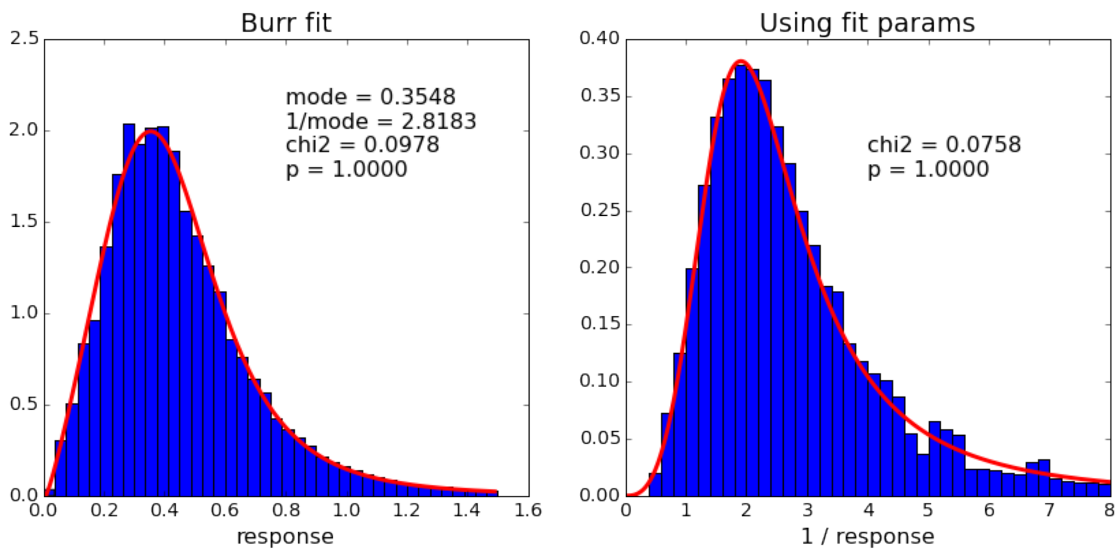
```
In [177]: apply_fit_to_inverse(rsp, scipy.stats.exponnorm, 'Exponnorm')
```

```
(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
```

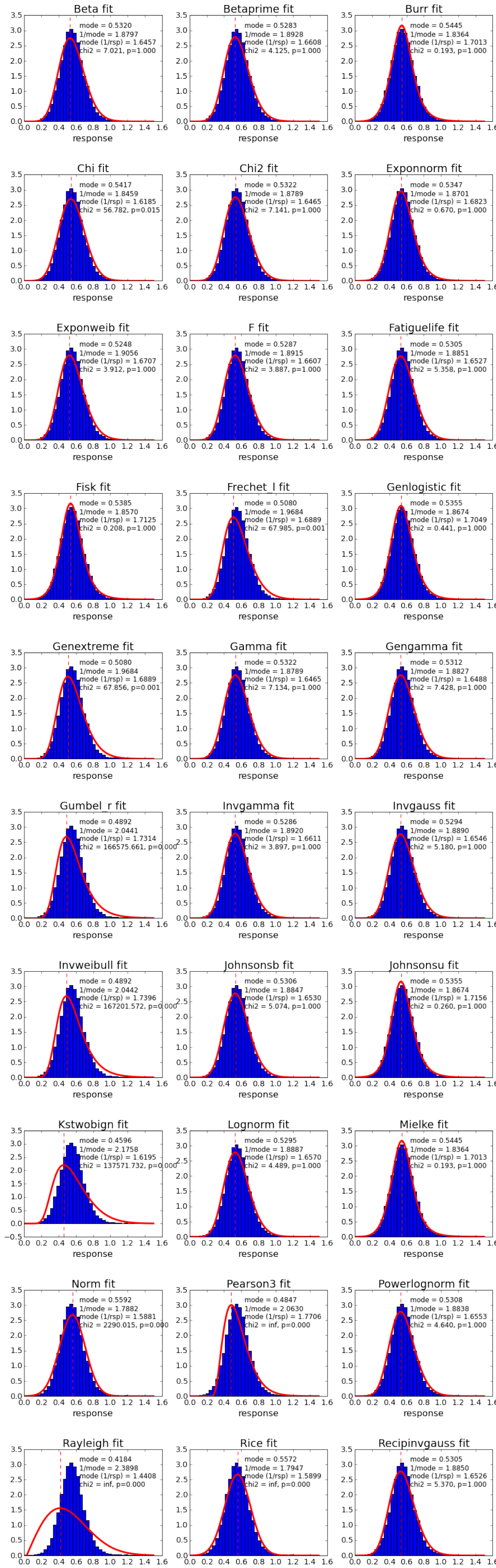
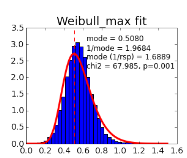


1.4 Higher ptRef bin (102 - 106 GeV)

```
In [197]: rspHigh_fit_fns = deepcopy(fit_fns)
          plot_multiple_fits(rspHigh, rspHigh_fit_fns, 'response', [0, 1.5])
```

```
Doing Beta
(27.627921563883277, 1876.4947154228075, -0.21949602920468314, 53.681175882775648)
(27.627921563883277, 1876.4947154228075) -0.219496029205 53.6811758828 0.531989729756 1.64571558527 7.02
Doing Betaprime
(108.07417236509849, 97.26478080922584, -0.48266578920599124, 0.9278029051891068)
(108.07417236509849, 97.26478080922584) -0.482665789206 0.927802905189 0.528314198499 1.66081967842 4.12
Doing Burr
(9.735131199234953, 0.71372751371132259, -0.13152277821839148, 0.71833840441299757)
(9.735131199234953, 0.71372751371132259) -0.131522778218 0.718338404413 0.544543818258 1.70133703157 0.2
Doing Chi
(9.9014907005457964, -0.087650378980587432, 0.21095450730176082)
(9.9014907005457964,) -0.0876503789806 0.210954507302 0.541740110298 1.61845798211 56.7816311629 0.01509
Doing Chi2
(58.999349670987954, -0.23743325161347356, 0.01350278292263286)
(58.999349670987954,) -0.237433251613 0.0135027829226 0.532216585349 1.64651668568 7.14057720769 0.99999
Doing Exponnorm
(0.84875454831649444, 0.4651753685823341, 0.11080386335341705)
(0.84875454831649444,) 0.465175368582 0.110803863353 0.53473075832 1.68230876812 0.669922284801 1.0
Doing Exponweib
(18.677146953242648, 2.5402414260169284, -0.52749108584235449, 0.67132979941812954)
(18.677146953242648, 2.5402414260169284) -0.527491085842 0.671329799418 0.524774320575 1.67071831434 3.9
Doing F
(3015.145217184514, 176.41932383621301, -0.7648533186650831, 1.3090676678630686)
(3015.145217184514, 176.41932383621301) -0.764853318665 1.30906766786 0.528681772423 1.66070008157 3.88
Doing Fatiguelife
(0.13245928010058697, -0.54577019527802162, 1.0953811583443289)
(0.13245928010058697,) -0.545770195278 1.09538115834 0.530477393548 1.65268964238 5.35768910301 0.999999
Doing Fisk
(13.752980814951648, -0.54142604534894523, 1.0914255029700914)
(13.752980814951648,) -0.541426045349 1.09142550297 0.53849940814 1.71249415388 0.207734287854 1.0
Doing Frechet_l
(11.396536452996163, 2.0470833208706631, 1.5515028407070162)
(11.396536452996163,) 2.04708332087 1.55150284071 0.50803278074 1.68894109387 67.9845310818 0.00100016
Doing Genlogistic
(1.6896939157016588, 0.48692716072634656, 0.092633280099444743)
(1.6896939157016588,) 0.486927160726 0.0926332800994 0.535517698461 1.70487071855 0.441079471134 1.0
Doing Genextreme
(0.087757663143128301, 0.49558013461916495, 0.13614173605672283)
(0.087757663143128301,) 0.495580134619 0.136141736057 0.508034494429 1.68892506241 67.8564287802 0.0010
Doing Gamma
(29.50223441983999, -0.23750837835834399, 0.027005804646788641)
(29.50223441983999,) -0.237508378358 0.0270058046468 0.532217367013 1.64649557408 7.13443525006 0.99999
Doing Gengamma
(24.524144135097885, 1.0479865539780864, -0.20206779918379913, 0.035951332692764473)
(24.524144135097885, 1.0479865539780864) -0.202067799184 0.0359513326928 0.531151748474 1.64881790122 7
Doing Gumbel_r
(0.48920459906352026, 0.13716774035923804)
None 0.489204599064 0.137167740359 0.489204591844 1.73135495589 166575.66067 0.0
Doing Invgamma
(87.824545763628791, -0.79947581302782034, 117.96171649077121)
```

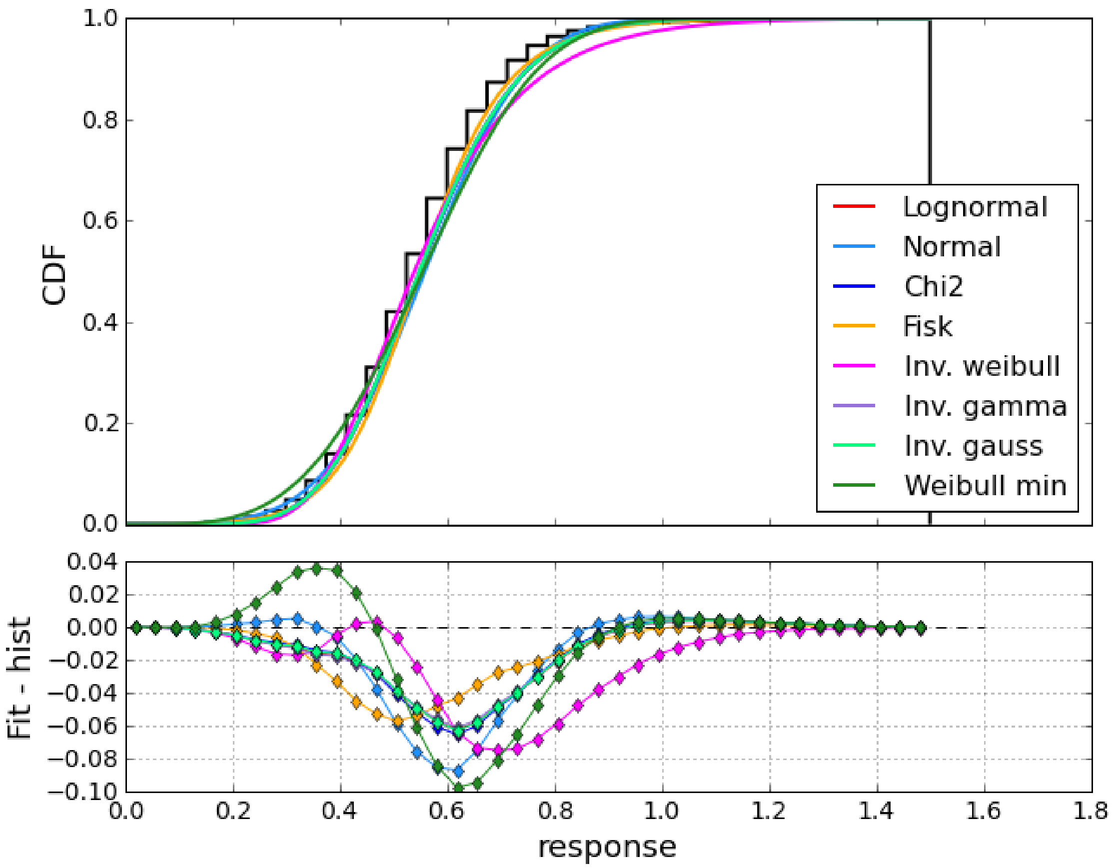
```
(87.824545763628791,) -0.799475813028 117.961716491 0.528554772027 1.6610818197 3.89720751422 0.99999999
Doing Invgauss
(0.018560309743589215, -0.51990259328985067, 58.128910950768386)
(0.018560309743589215,) -0.51990259329 58.1289109508 0.529369214926 1.65462095491 5.18004555049 0.999999
Doing Invweibull
(597849143.99745035, -82001284.16489476, 82001284.654092506)
(597849143.99745035,) -82001284.1649 82001284.6541 0.489197778217 1.73962808319 167201.572367 0.0
Doing Johnsonsb
(17.153369643822209, 6.5273814486875184, -0.4648807332916543, 15.061307704810281)
(17.153369643822209, 6.5273814486875184) -0.464880733292 15.0613077048 0.530600028911 1.65298129076 5.07
Doing Johnsonsu
(-0.54095407043069543, 1.9571624247160149, 0.48238483286398659, 0.23956296743562344)
(-0.54095407043069543, 1.9571624247160149) 0.482384832864 0.239562967436 0.535489731317 1.71559037992 0
Doing Kstwobign
(-0.10337642121292279, 0.76548299293354105)
None -0.103376421213 0.765482992934 0.459611594596 1.61954588856 137571.732163 0.0
Doing Lognorm
(0.13759441782271659, -0.50162291882089771, 1.050787176716043)
(0.13759441782271659,) -0.501622918821 1.05078717672 0.529457646159 1.65703591617 4.48864525958 0.999999
Doing Mielke
(6.9481119328920062, 9.7351044839172687, -0.13151612735908236, 0.71833325148921623)
(6.9481119328920062, 9.7351044839172687) -0.131516127359 0.718333251489 0.544544289836 1.70133622465 0.
Doing Norm
(0.55922216998743335, 0.14896318771273687)
None 0.559222169987 0.148963187713 0.559222108879 1.58805751371 2290.01488106 0.0
Doing Pearson3
(1.0, 0.55922216998743335, 0.14896318771273687)
(1.0,) 0.559222169987 0.148963187713 0.484740569064 1.77057999666 inf 0.0
Doing Powerlognorm
(1.4768148753397319, 0.17702164660492398, -0.3654325567034698, 0.96814320112278573)
(1.4768148753397319, 0.17702164660492398) -0.365432556703 0.968143201123 0.530831748343 1.65529199658 4
Doing Rayleigh
(0.029027127350346574, 0.38941393800215679)
None 0.0290271273503 0.389413938002 0.418441165582 1.44083456834 inf 0.0
Doing Rice
(3.3423554762911545, 0.024972238722121863, 0.15282305650166428)
(3.3423554762911545,) 0.0249722387221 0.152823056502 0.557184111905 1.58992030316 inf 0.0
Doing Recipinvgauss
(0.017692287490253915, -0.54364881947190669, 0.019173105844558292)
(0.017692287490253915,) -0.543648819472 0.0191731058446 0.530505540068 1.65261089538 5.37011905148 0.999
Doing Weibull_max
(11.396536452996163, 2.0470833208706631, 1.5515028407070162)
(11.396536452996163,) 2.04708332087 1.55150284071 0.50803278074 1.68894109387 67.9845310818 0.001000166
```


Beta fit
mode = 0.5320
1/mode = 1.8797
mode (1/rsp) = 1.6457
chi2 = 7.021, p=1.000
Betaprime fit
mode = 0.5283
1/mode = 1.8928
mode (1/rsp) = 1.6608
chi2 = 4.125, p=1.000
Burr fit
mode = 0.5445
1/mode = 1.8364
mode (1/rsp) = 1.7013
chi2 = 0.193, p=1.000
Chi fit
mode = 0.5417
1/mode = 1.8459
mode (1/rsp) = 1.6185
chi2 = 56.782, p=0.015
Chi2 fit
mode = 0.5322
1/mode = 1.8789
mode (1/rsp) = 1.6465
chi2 = 7.141, p=1.000
Exponnorm fit
mode = 0.5347
1/mode = 1.8701
mode (1/rsp) = 1.6823
chi2 = 0.670, p=1.000
Exponweib fit
mode = 0.5248
1/mode = 1.9056
mode (1/rsp) = 1.6707
chi2 = 3.912, p=1.000
F fit
mode = 0.5287
1/mode = 1.8915
mode (1/rsp) = 1.6607
chi2 = 3.887, p=1.000
Fatiguelife fit
mode = 0.5305
1/mode = 1.8851
mode (1/rsp) = 1.6527
chi2 = 5.358, p=1.000
Fisk fit
mode = 0.5385
1/mode = 1.8570
mode (1/rsp) = 1.7125
chi2 = 0.208, p=1.000
Frechet_l fit
mode = 0.5080
1/mode = 1.9684
mode (1/rsp) = 1.6889
chi2 = 67.985, p=0.001
Genlogistic fit
mode = 0.5355
1/mode = 1.8674
mode (1/rsp) = 1.7049
chi2 = 0.441, p=1.000
Genextreme fit
mode = 0.5080
1/mode = 1.9684
mode (1/rsp) = 1.6889
chi2 = 67.856, p=0.001
Gamma fit
mode = 0.5322
1/mode = 1.8789
mode (1/rsp) = 1.6465
chi2 = 7.134, p=1.000
Gengamma fit
mode = 0.5312
1/mode = 1.8827
mode (1/rsp) = 1.6488
chi2 = 7.428, p=1.000
Gumbel_r fit
mode = 0.4892
1/mode = 2.0441
mode (1/rsp) = 1.7314
chi2 = 166575.661, p=0.000
Invgamma fit
mode = 0.5286
1/mode = 1.8920
mode (1/rsp) = 1.6611
chi2 = 3.897, p=1.000
Invgauss fit
mode = 0.5294
1/mode = 1.8890
mode (1/rsp) = 1.6546
chi2 = 5.180, p=1.000
Invweibull fit
mode = 0.4892
1/mode = 2.0442
mode (1/rsp) = 1.7396
chi2 = 167201.572, p=0.000
Johnsonsb fit
mode = 0.5306
1/mode = 1.8847
mode (1/rsp) = 1.6530
chi2 = 5.074, p=1.000
Johnsonsu fit
mode = 0.5355
1/mode = 1.8674
mode (1/rsp) = 1.7156
chi2 = 0.260, p=1.000
Kstwobign fit
mode = 0.4596
1/mode = 2.1758
mode (1/rsp) = 1.6195
chi2 = 137571.732, p=0.000
Lognorm fit
mode = 0.5295
1/mode = 1.8887
mode (1/rsp) = 1.6570
chi2 = 4.489, p=1.000
Mielke fit
mode = 0.5445
1/mode = 1.8364
mode (1/rsp) = 1.7013
chi2 = 0.193, p=1.000
Norm fit
mode = 0.5592
1/mode = 1.7882
mode (1/rsp) = 1.5881
chi2 = 2290.015, p=0.000
Pearson3 fit
mode = 0.4847
1/mode = 2.0630
mode (1/rsp) = 1.7706
chi2 = inf, p=0.000
Powerlognorm fit
mode = 0.5308
1/mode = 1.8838
mode (1/rsp) = 1.6553
chi2 = 4.640, p=1.000
Rayleigh fit
mode = 0.4184
1/mode = 2.3898
mode (1/rsp) = 1.4408
chi2 = inf, p=0.000
Rice fit
mode = 0.5572
1/mode = 1.7947
mode (1/rsp) = 1.5899
chi2 = inf, p=0.000
Recipinvgauss fit
mode = 0.5305
1/mode = 1.8850
mode (1/rsp) = 1.6526
chi2 = 5.370, p=1.000
response

Weibull_max fit
mode = 0.5080
1/mode = 1.9684
mode (1/rsp) = 1.6889
chi2 = 67.985, p=0.001
response

```
In [198]: print_ordered_fit_fn(rspHigh_fit_fns)
```

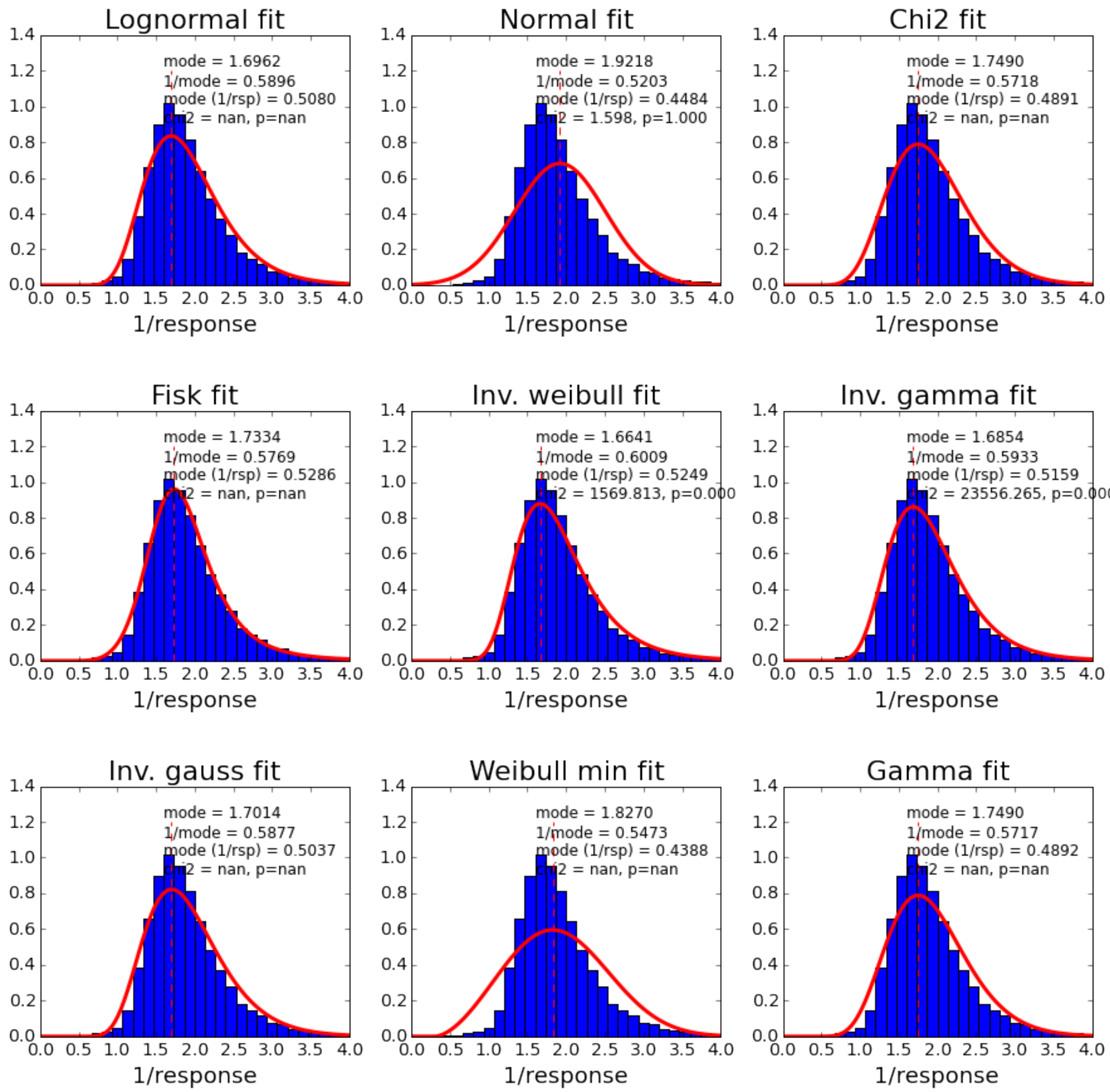
```
Burr 0.193231977087 1.0
Mielke 0.193234079093 1.0
Fisk 0.207734287854 1.0
Johnsonsu 0.260199190997 1.0
Genlogistic 0.441079471134 1.0
Exponnorm 0.669922284801 1.0
F 3.88720434921 0.999999999988
Invgamma 3.89720751422 0.999999999996
Exponweib 3.9117761173 0.999999999987
Betaprime 4.12541452624 0.99999999997
Lognorm 4.48864525958 0.999999999961
Powerlognorm 4.63992428165 0.999999999814
Johnsonsb 5.07359780883 0.999999999274
Invgauss 5.18004555049 0.999999999627
Fatiguelife 5.35768910301 0.999999999371
Recipinvgauss 5.37011905148 0.999999999348
Beta 7.0212762459 0.999999914071
Gamma 7.13443525006 0.999999952622
Chi2 7.14057720769 0.999999952021
Gengamma 7.42810393804 0.999999809608
Chi 56.7816311629 0.0150986400659
Genextreme 67.8564287802 0.00103430458066
Frechet_l 67.9845310818 0.00100016691835
Weibull_max 67.9845310818 0.00100016691835
Norm 2290.01488106 0.0
Kstwobign 137571.732163 0.0
Gumbel_r 166575.66067 0.0
Invweibull 167201.572367 0.0
Pearson3 inf 0.0
Rayleigh inf 0.0
Rice inf 0.0
```

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

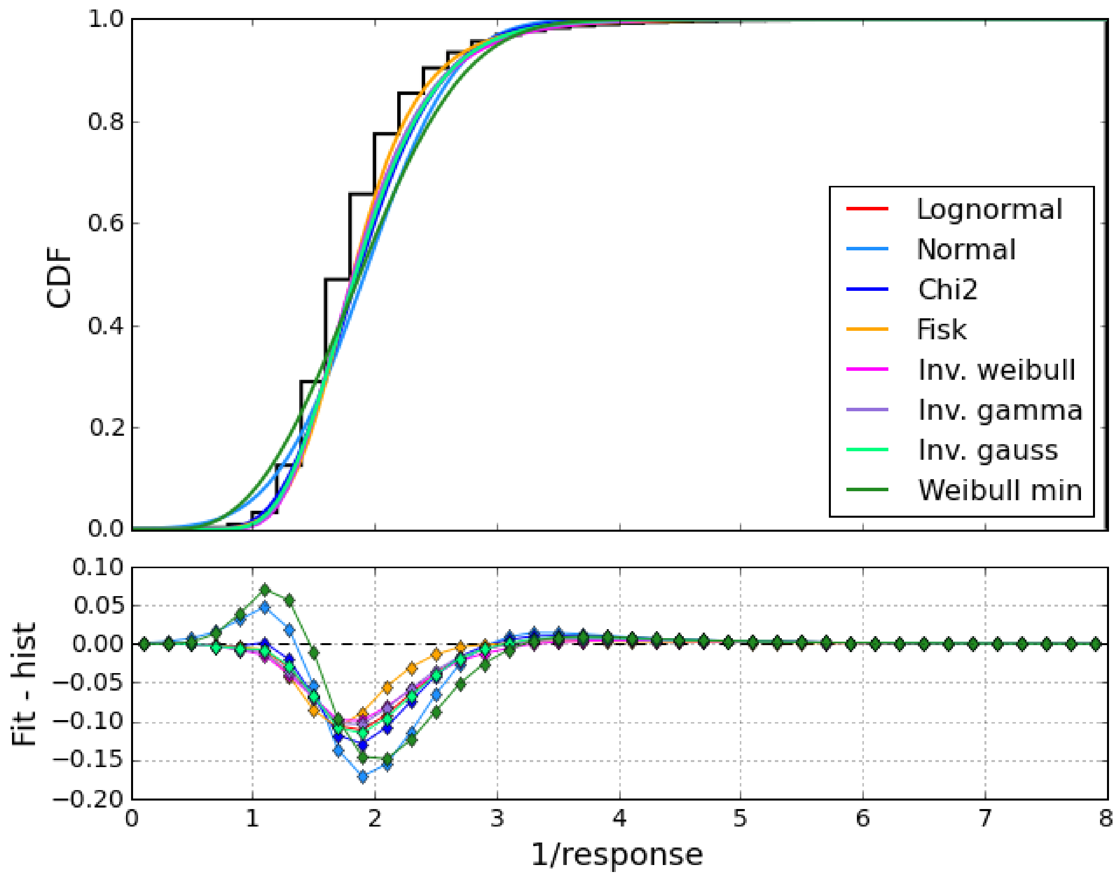



```
In [94]: rspHighInv_fit_fns = deepcopy(fit_fns)
         plot_multiple_fits(rspHighInv, rspHighInv_fit_fns, '1/response', [0, 4])
```

```
/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.
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
```

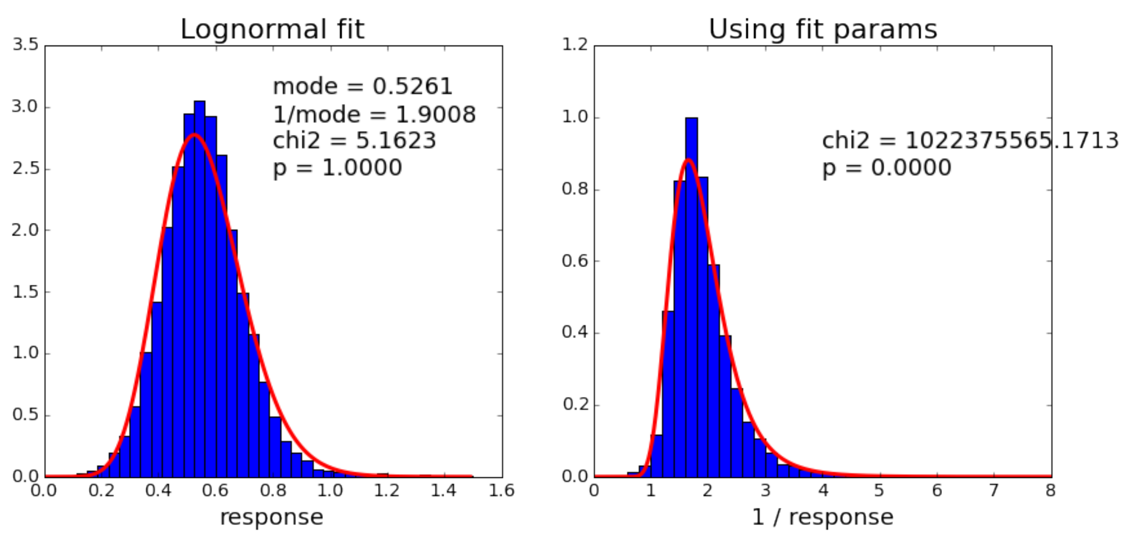



```
In [95]: plot_cdf(rspHighInv, rspHighInv_fit_fns, '1/response', [0, 8])
```



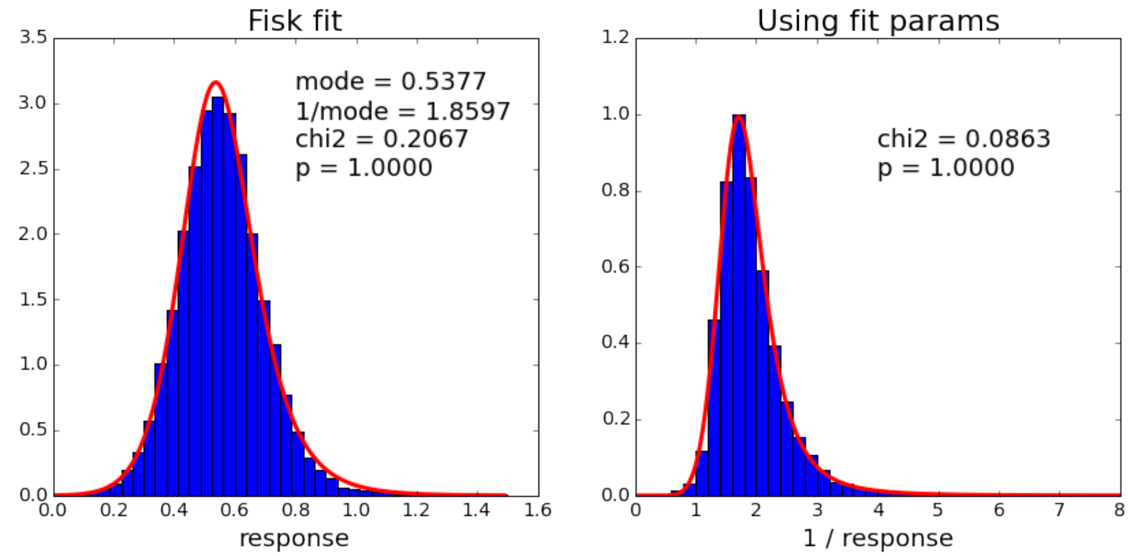

```
In [72]: apply_fit_to_inverse(rspHigh, scipy.stats.lognorm, 'Lognormal')
```

```
0.154298584152 -0.395180479058 0.943463158564
```



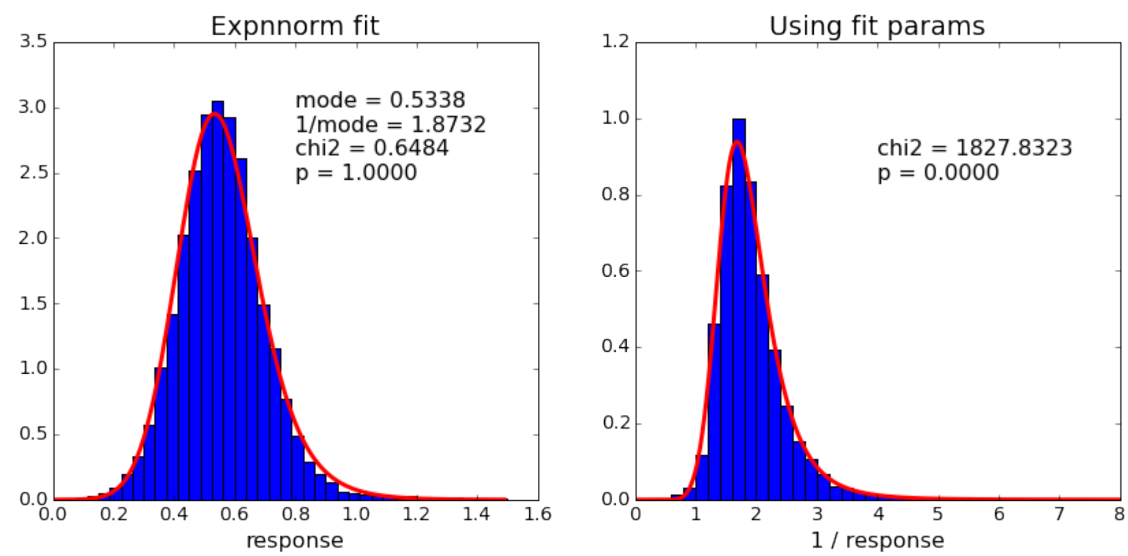
```
In [73]: apply_fit_to_inverse(rspHigh, scipy.stats.fisk, 'Fisk')
```

13.0522913071 -0.488558211959 1.03842389756



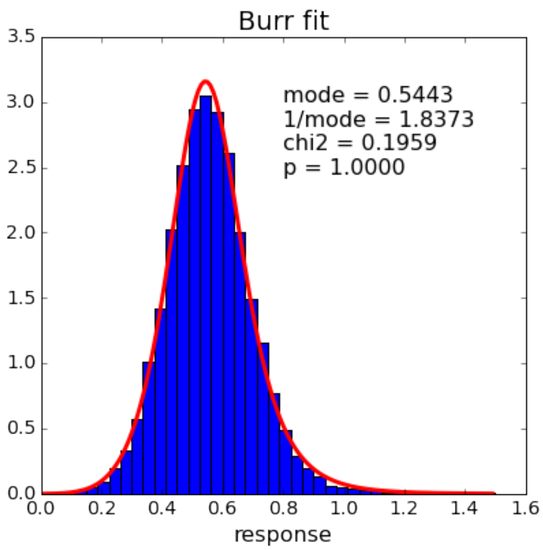

```
In [179]: apply_fit_to_inverse(rspHigh, scipy.stats.exponnorm, 'Expnnorm')
```

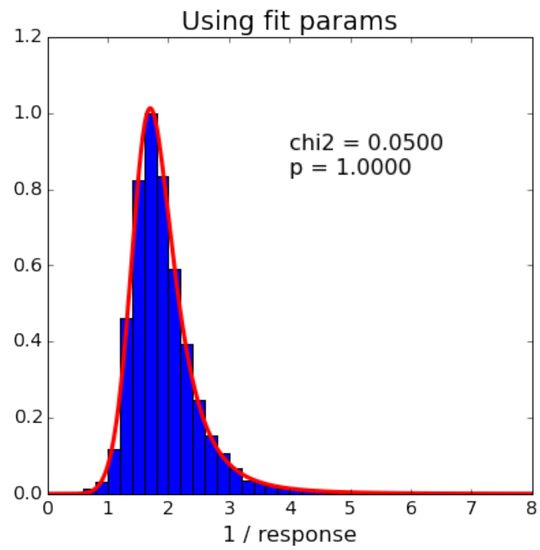
(0.87557410318578643,) 0.463330557512 0.110058418106




```
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)
```

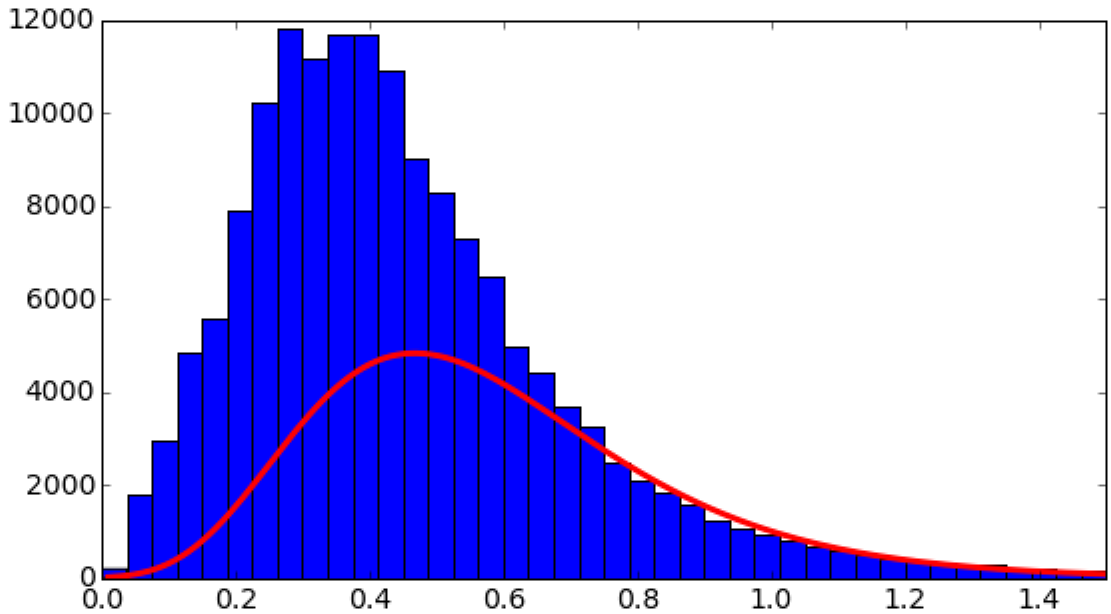
Out[100]:
```
(array([ 0.2,  0.4,  0.6,  0.8]),
 array([ 0.00089758,  0.05953092,  0.28407908,  0.57780375]))
```

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
```

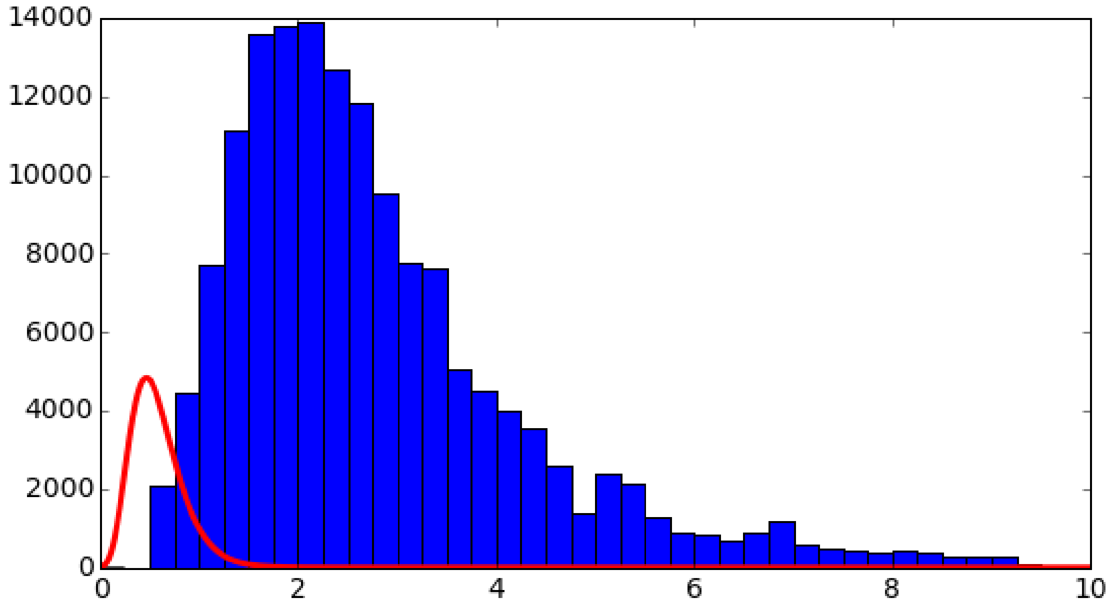
```
                m=widgets.FloatSlider(min=0, max=5, step=0.01, value=0.63, continuous_update=False),
                theta=widgets.FloatSlider(min=-10, max=10, step=0.01, value=-0.23, continuous_update=
                sigma=widgets.FloatSlider(min=0, max=10, step=0.01, value=0.32, continuous_update=Fal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

Out[102]: <function __main__.plot_hist_fn>



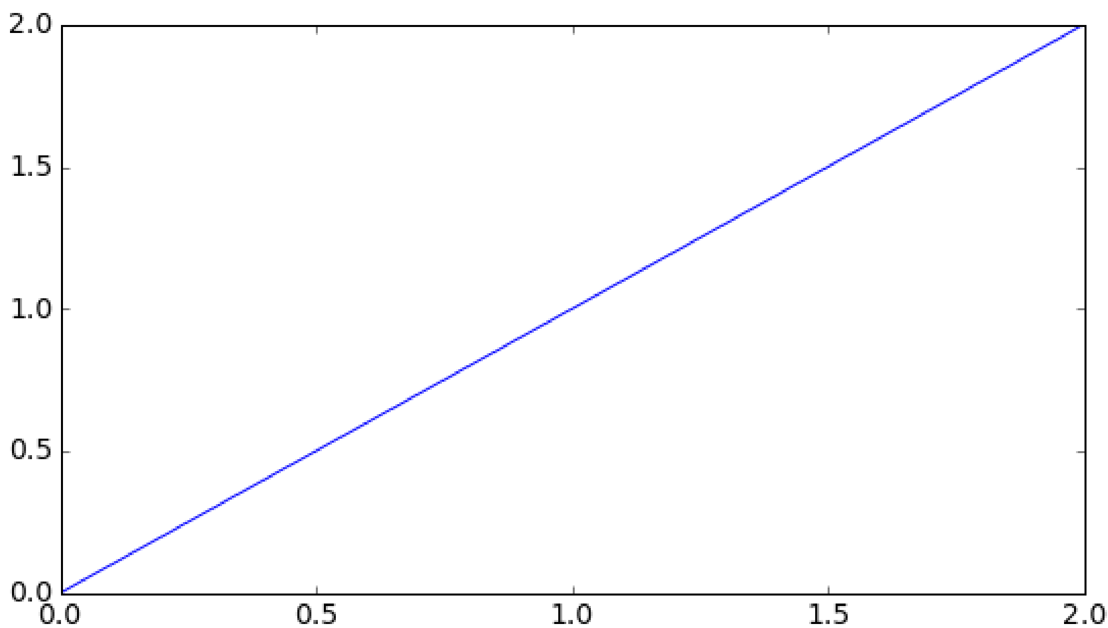

```
In [103]: interact(plot_hist_fn, hist_data=fixed(rspInv), bins=fixed(40), xlim=fixed([0, 10]),
                x=fixed(x),
                fn=fixed(my_lognorm),
                N=widgets.FloatSlider(min=1, max=50000, step=50, value=5851, continuous_update=True)
                m=widgets.FloatSlider(min=0,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