Title:	Outlier detection in small samples using pull-clipping		
Date:	2016/11/03	**Issue:**	Draft 0.1
Reference:	EUCL-IPN-TN-8-003		
Custodian:	Yannick Copin (y.copin@ipnl.in2p3.fr)		

Authors:		**Date:**	**Signature:**
	Yannick Copin *(IPNL)*	2016/11/03	
Contributors:			
Approved by:			
Authorized by:			

Document version tracking

Issue	Date	Page	Description of changes	Comments
0.1	2016/11/03	11	First release	First draft.

Table of contents

1 Purpose 3

2 Scope 3

3 Applicable & Reference documents 3
3.1 Applicable documents 3
3.2 Reference documents 3

4 Acronyms 3

5 Outlier detection in small samples using pull-clipping 3
5.1 Introduction 4
5.2 Outlier detection 5
5.2.1 σ-clipping (Grubbs' tests for outliers) 5
5.2.2 Pull-clipping 6
5.3 Implementation and numerical experiments 7
5.3.1 Input population 7
5.3.2 Median statistic 10
5.3.3 Outlier detection (upper one-tailed) 10
5.3.4 Analysis 15
5.4 Conclusions 19
5.5 Appendix 20
5.5.1 Clip study 20

1 Purpose

We present and tests different algorithms to compute the mean of very small samples ($n = 4$) in presence of a small fraction of outliers. The favorite choice appears to be the inverse-variance weighted mean of pull-clipped values, both more robust and statistically efficient (from 10% to 40%) than the median.

2 Scope

Spectra combination in OU-SIR.

Applicable work packages:

1. *Spectra Combination* Wp-4-3-07-5100 Simulation_140425

3 Applicable & Reference documents

3.1 Applicable documents

RD		Ref.	Date

3.2 Reference documents

RD		Ref.	Date

4 Acronyms

ML	Maximum Likelihood
PDF	Probability Density Function
SL	Significance Level
SNR	Signal-to-Noise Ratio
TBC	To Be Completed

Table of contents

5 Outlier detection in small samples using pull-clipping

```
In [1]: %matplotlib inline

        from __future__ import division, print_function

        import numpy as N
        import scipy.stats as SS

        import astropy.stats as AS
        import astropy.table as AT

        import matplotlib.pyplot as P

        from combine import *

        N.random.seed(12345)
```

5.1 Introduction

Consider a sample of independent (possibly-fallacious) measurements $\{x_i\}$ of the same quantity X, with presumably normal-errors $\{\sigma_i\}$. One may want to combine these measurements into a single sample mean estimate. Typical user case is the combination of multiple images or spectra in presence of cosmic rays.

In absence of outliers, the Maximum Likelihood Estimate of the mean of measurements $\{x_i\}$ normally distributed with standard deviation $\{\sigma_i\}$ is the inverse-variance weighted mean:

$$\mu_w = \frac{1}{w}\sum_i w_i x_i \quad \text{with} \quad w_i = \sigma_i^{-2} \quad \text{and} \quad w = \sum_i w_i \tag{1}$$

$$\sigma_w = w^{-1/2} \tag{2}$$

Unfortunately, the (weighted) mean is linearly sensitive to outliers: if a value is off by a factory η, the mean is off by a factor directly $\propto \eta$.

In presence of outliers, it is traditional to apply the median statistic, known to be both robust (breakdown point of 50%: up to half of the points can be infinite outliers) and easy to compute:

$$m = \mathrm{med}(\{x_i\}) \tag{3}$$

$$\sigma_m = \left(\eta_n \langle 1/\sigma_i \rangle_i^2\right)^{-1/2} \tag{4}$$

$$\eta_n = \frac{2}{\pi} \times \begin{cases} (n + \pi - 2) & n \text{ even} \\ (n + \pi/2 - 1) & n \text{ odd} \end{cases} \tag{5}$$

However, the median has major drawbacks, specially in the case of very small samples:

- The median is known to be significantly less *efficient* than standard mean, with a variance increased by a factor $\pi/2 \times n/(n-1)$, ranging from $\pi/2 \sim 1.6$ for large normal samples ($n \gg 1$) to a factor 2 for $n = 4$ samples (in the noiseless case). This typically corresponds to an equivalent *waste* of effective exposure time needed to reach a given signal-to-noise ratio on the combined measurement.
- The median does not make any standard use of measured errors, treating all measurements as equally accurate.
- Furthermore, it is generally the case that there are much less than 50% of outliers in the sample, and the usage of the median is then overdone.

Other robust location estimators (e.g. trimmed/winsorized means) could be of interest in the general case, but their usage is not adapted to very small samples.

Note: We explicitely consider here samples without intrinsic dispersion, i.e. where all the observed scatter of $\{x_i\}$ can be statistically explained by measurement errors $\{\sigma_i\}$. In case of non-null instrinsic dispersion σ, the proper weighting is $w_i = (\sigma^2 + \sigma_i^2)^{-1}$.

For very small sample sizes, it therefore appears the optimally-weighted average on an outlier-clipped sample is the best alternative for both a robust and accurate estimate of the sample mean.

5.2 Outlier detection

5.2.1 σ-clipping (Grubbs' tests for outliers)

A standard way to detect a *single* outlier in a normally-distributed sample is the *Grubbs' test* (see e.g. NIST/SEMATECH e-Handbook of Statistical Methods):

1. estimate the mean $\bar{x}$ and standard deviation s for the sample;
2. compute the z-score for all measurement: $z_i = (x_i - \bar{x})/s$;
3. the measurement with maximal (resp. minimal) z-score is declared an upper (resp. lower) outlier if

$$z > z_{\max} = \frac{n-1}{\sqrt{n}}\sqrt{\frac{t^2_{\alpha/2n,n-2}}{n-2+t^2_{\alpha/2n,n-2}}}$$

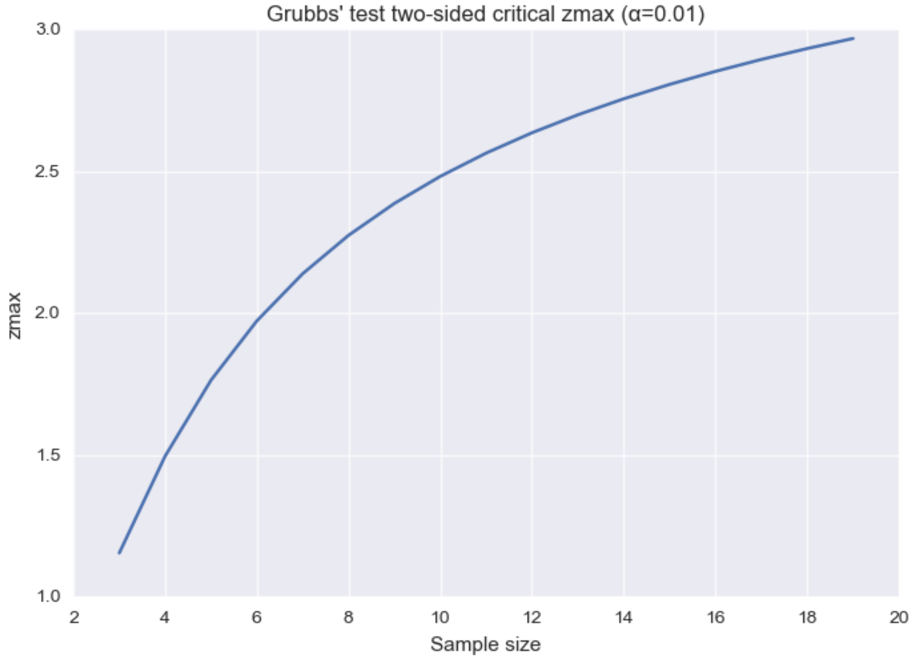
   where $t_{\alpha/2n,n-2}$ is the critical value of the t-distribution with $(n-2)$ degrees of freedom and a significance level of $\alpha/(2N)$ (for a two-sided test).

```
In [2]: t = N.array([199.31, 199.53, 200.19, 200.82, 201.92, 201.95, 202.18, 245.57])
        grubbs_test(t, side=+1, alpha=0.05, modified=False, verbose=True)
```

```
H0:  there are no outliers in the data
Ha:  the maximum value is an outlier

Test statistic:      G = [ 2.46876461]
Significance level:   = 0.05
Critical value for an upper one-tailed test: G_max = 2.03165200155
Critical region:     Reject H0 if G > G_max


Out[2]: array([False, False, False, False, False, False, False,  True], dtype=bool)
```

Grubbs' test for outliers actually closely relates to the intuitive *σ-clipping* but with a statistically-founded clip value $z_{\max}$.

```
In [3]: ns = N.arange(3, 20)
        fig, ax = P.subplots()
        ax.plot(ns, [ grubbs_gmax(_, alpha=1e-2) for _ in ns ], 'b-')
        ax.set(xlabel='Sample size', ylabel='zmax', title=u"Grubbs' test two-sided critical zmax (=0.01)");
```



Grubbs' test can be generalized in two different ways:

- by using *robust* sample estimates, median and (normalized) Median Absolute Deviation, to compute the *modified* z-score;
- by using (inverse-variance) *weighted* sample mean $\bar{x} = \mu_w$ for location and error estimate $s = \sigma_i$ for scale (since σ_i is by definition the estimate of the dispersion of x_i).

Note: Formally, because of potential screening effects (multiple outliers weaken the single-outlier test performance), it is supposedly not appropriate to apply the single-outlier Grubbs' test sequentially in order to detect multiple outliers. In the following application, we will still apply it sequentially until no more outlier is detected.

5.2.2 Pull-clipping

As traditionnaly defined, Grubbs' test has two drawbacks:

- all measurements, including a potential outlier, are equally considered in the estimate of the location and scale of the sample needed for the z-score,
- measurement errors $\{\sigma_i\}$ are not (fully) used in the traditional Grubbs' test for outliers.

The *pull statistic* is a seamingly more pertinent quantity than the z-score. For each measurement of the sample, the *pull* p_i is defined by:

$$p_i = \frac{x_i - \bar{x}_i}{\sqrt{\sigma_i^2 + \bar{\sigma}_i^2}}$$

where $\bar{x}_i$ and $\bar{\sigma}_i$ are the inverse-variance weighted average and its associated error of the sample *without* point i.

We therefore introduce the *pull-clipping*, a procedure similar to Grubbs' test for outliers but using the *pull* in place of the z-score. We did not try to derive a statistically-founded clipping value $p_{\max}$, which will have to be estimated from numerical experimentations.

If errors $\{\sigma_i\}$ are correct and fully represent the observed dispersion of measurements $\{x_i\}$, the pull have a normal distribution $\mathcal{N}(0,1)$.

```
In [4]: dt = SS.lognorm(0.25).rvs(size=4)   # dt ~ 1
        t = SS.norm(loc=0, scale=dt).rvs()  # t ~ 0 ± 1
        t[0] += 10                          # 10-sigma outlier

        table = AT.Table([t, dt, zscore(t), zscore(t, modified=True), zscore(t, dt), pull(t, dt)],
                         names=('t', 'dt', 'z-score', 'robust', 'weighted', 'pull'))
        table.pprint(max_width=-1)
```

```
       t              dt           z-score          robust        weighted         pull
--------------- -------------- --------------- --------------- -------------- --------------
  11.8677088943 0.950110566974   1.48954538014   9.92881633045  9.03535249725  10.4022034912
  1.57064572445  1.12719904516 -0.330674762768  0.600381602355 -1.51923415231  -1.6720597059
0.0815934305529 0.878218655002 -0.593895735048 -0.748597898037 -3.64548345851 -4.31842958085
 0.245199930885 0.870286704615 -0.564974882325 -0.600381602355 -3.49071755701 -4.15044712671
```

5.3 Implementation and numerical experiments

We develop here a numerical experiment supposed to mimick the spectrum combination cases when some of the pixel are affected by cosmic rays residuals.

5.3.1 Input population

Define a population of `m = 10000` samples of `n = 4` measurements (e.g. `n` spectra of `m` pixels to be combined). Each point have a distribution $\mathcal{N}(\mu = 0, \sigma^2 \sim 1)$, the actual errors being distributed around 1 with a log-normal distribution $\ln\mathcal{N}(\mu = 0, \sigma^2 = 0.25^2)$.

```
In [5]: n = 4                       # Nb of points per sample
        m = 10000                   # Nb of samples
        size = (m, n)
        print("{} realizations of {}-samples".format(m, n))
```

```
10000 realizations of 4-samples
```

```
In [6]: shape = 0.25
        dx_dist = SS.lognorm(shape)  # Log-normal distribution
```

Each value has a uniform probability $\epsilon = 5\%$ of being an outlier:

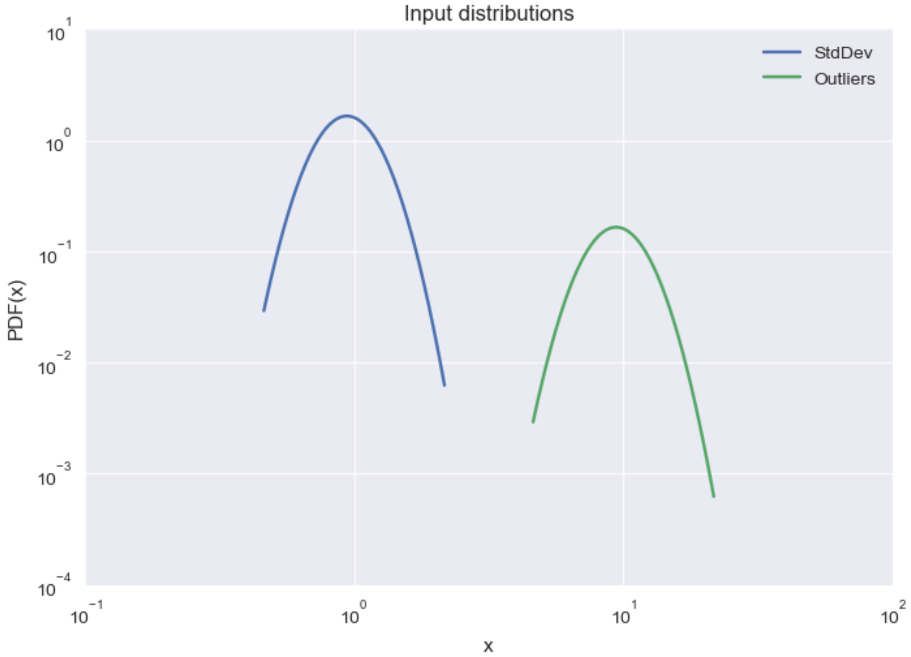
```
In [7]: eps = 0.05
        print("Probability of outliers: {:.1%}".format(eps))

        outliers = N.random.rand(m, n) <= eps  # Outlier mask

        noutliers = outliers.sum(axis=-1)  # Nb of outliers in each sample

        rows = [ (i, SS.binom.pmf(i, n, eps), len(noutliers[noutliers == i]), len(noutliers[noutliers == i])
                 for i in range(n + 1) ]
        table = AT.Table(rows=rows, names=('Outliers', 'Probability', '# of cases', 'Fraction'))
        print(table)
```

```
Probability of outliers: 5.0%
Outliers Probability # of cases Fraction
-------- ----------- ---------- --------
       0  0.81450625       8177   0.8177
       1    0.171475       1691   0.1691
       2   0.0135375        128   0.0128
       3    0.000475          4   0.0004
       4    6.25e-06          0      0.0
```

Outliers follow a log-normal distribution $\ln \mathcal{N}(\mu = \ln(\texttt{nsig} = 10), \sigma^2 = 0.25^2)$:

```
In [8]: nsig = 10
        outliers_dist = SS.lognorm(shape, scale=nsig)
```

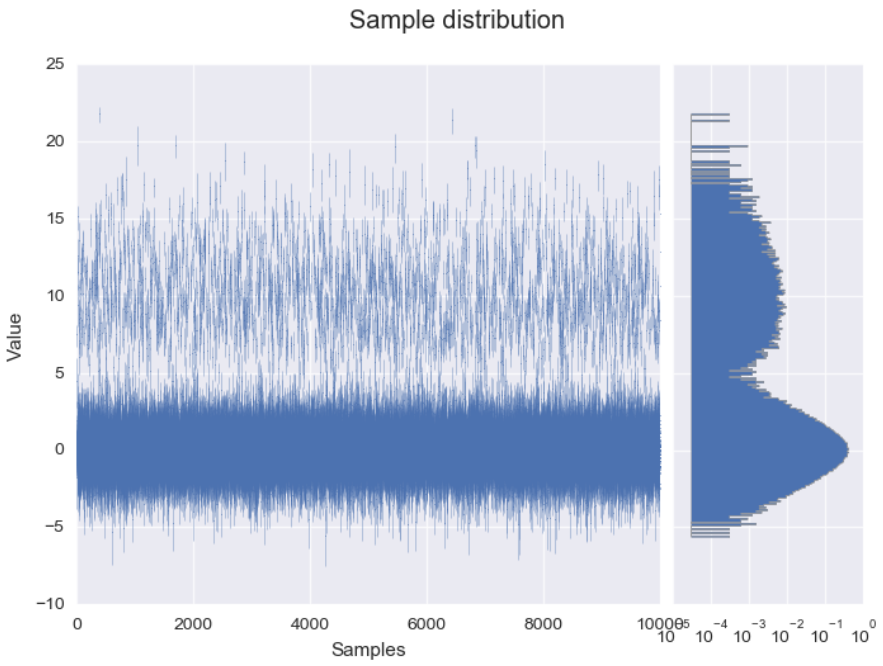
```
In [9]: ax = plot_rv(dx_dist, label='StdDev')
        ax.set(xscale='log', yscale='log')
        ax = plot_rv(outliers_dist, label='Outliers', ax=ax)
        ax.legend(fontsize='small')
        ax.set_title("Input distributions");
```



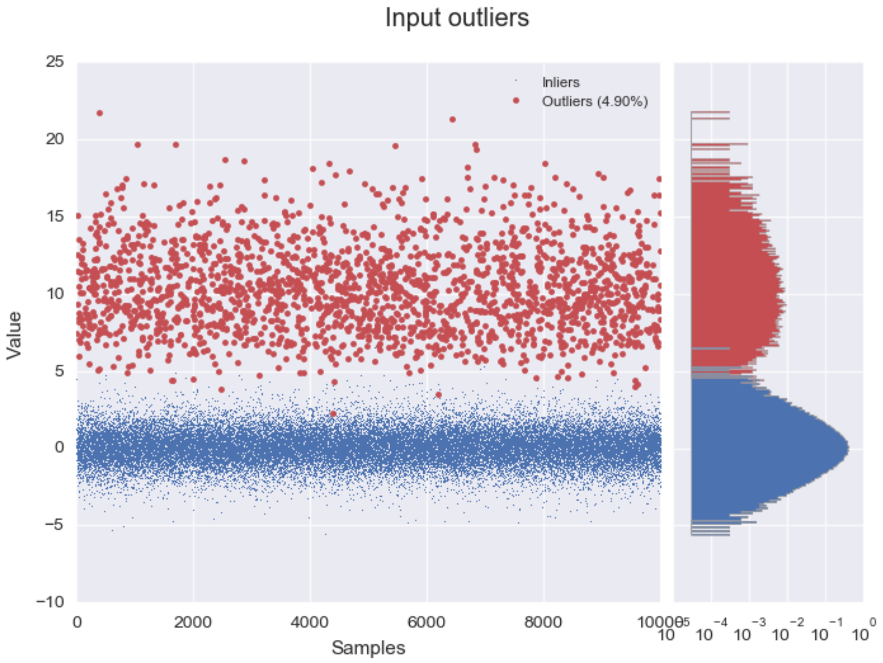
Note: We assume here the outlying process does not significantly affect the measurement variance.

```
In [10]: dx = dx_dist.rvs(size=size)
         x = N.random.normal(loc=0, scale=dx, size=size)  # Pure gaussian errors
         x += N.where(outliers, outliers_dist.rvs(size=size), 0)  # Add outliers
```

```
In [11]: fig = plot_samples(x, dx)
         fig.suptitle("Sample distribution", fontsize='large');
```



```
In [12]: fig = plot_samples(x, outliers=outliers)
         fig.suptitle("Input outliers", fontsize='large');
```



5.3.2 Median statistic

```
In [13]: med, dmed = sample_median(x, dx)       # Median (unweighted)
```

5.3.3 Outlier detection (upper one-tailed)

Since the outliers are supposed to mimick additive cosmic rays, we are only looking for *upper* outliers.

```
In [14]: side = +1  # Upper one-tailed outlier test
         alpha = 0.01
         zmax = grubbs_gmax(n, alpha, one_sided=True)
         print("Grubb's critical value for an upper one-tailed test for outliers: zmax={}".format(zmax))
```

```
Grubb's critical value for an upper one-tailed test for outliers: zmax=1.4925
```

σ-clipping (standard Grubbs' test)

```
In [15]: identified = outlier_clipping(x, dx, method='sigma', clip=zmax, side=side, verbose=True)
         detection_rates(outliers, identified, verbose=True);
```

```
Outlier sigma-clipping: upper one-tailed, clip=1.4925
  Iter #1: 666 outliers (total: 666)
  Iter #2: 0 outliers (total: 666)
Input outliers:      1959/40000 (4.90%)
Identified outliers: 666 (TP: 572, FP: 94, FN: 1387)
Sensitivity TPR=1-FNR: 29.20%
```

```
Specificity TNR=1-FPR: 99.75%
Precision PPV:         85.89%
Accuracy:              96.30%
Matthews Correlation Coefficient: 0.488
```

Standard Grubbs' test has a precision of 84% but a low sensitivity of 28% because of too many false negatives. This method will not be studied further.

nMAD-clipping (modified Grubbs's test)

```
In [16]: identified = outlier_clipping(x, dx, method='robust', clip=zmax, side=side, maxiter=1, verbose=True
         detection_rates(outliers, identified, verbose=True);
```

```
Outlier robust-clipping: upper one-tailed, clip=1.4925
  Iter #1: 4312 outliers (total: 4312)
Input outliers:      1959/40000 (4.90%)
Identified outliers: 4312 (TP: 1706, FP: 2606, FN: 253)
Sensitivity TPR=1-FNR: 87.09%
Specificity TNR=1-FPR: 93.15%
Precision PPV:         39.56%
Accuracy:              92.85%
Matthews Correlation Coefficient: 0.558
```

Modified Grubbs' test has a reasonable accuracy of 93% but a very low precision of 40% because of too many false positives (even when stopping outlier detection after a single iteration). A higher clipping value improves the overall efficiency (as estimated from Matthews Correlation Coefficient), but not to the level of the methods described below:

```
In [17]: identified = outlier_clipping(x, dx, method='robust', clip=3, side=side, maxiter=1, verbose=True)
         detection_rates(outliers, identified, verbose=True);
```

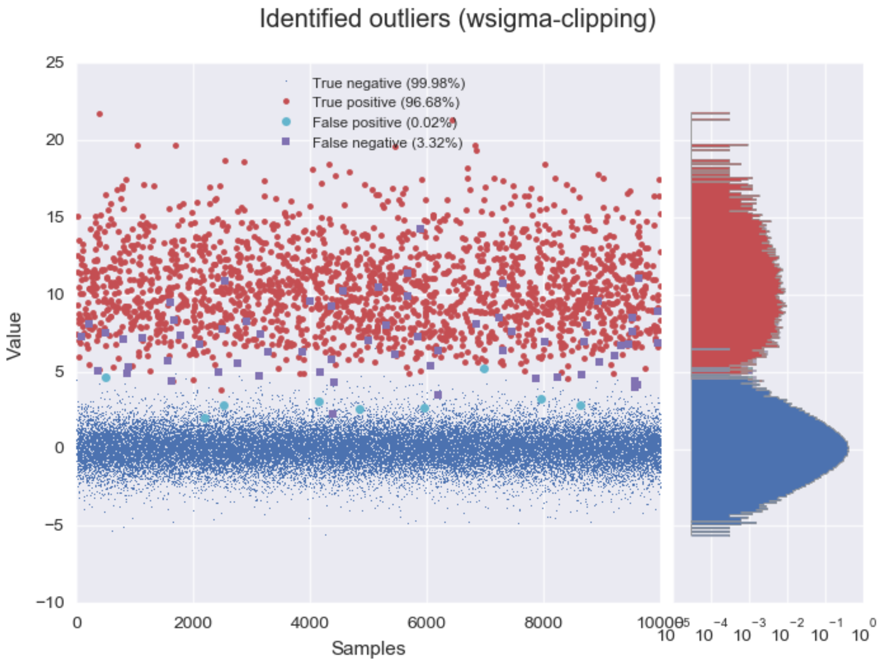
```
Outlier robust-clipping: upper one-tailed, clip=3
  Iter #1: 2546 outliers (total: 2546)
Input outliers:      1959/40000 (4.90%)
Identified outliers: 2546 (TP: 1591, FP: 955, FN: 368)
Sensitivity TPR=1-FNR: 81.21%
Specificity TNR=1-FPR: 97.49%
Precision PPV:         62.49%
Accuracy:              96.69%
Matthews Correlation Coefficient: 0.696
```

Weighted σ_w-clipping (weighted Grubbs's test) In absence of a theoritically justified clipping value, we use here numerically-optimized `clip=3` (see Appendix).

```
In [18]: clip = 3
         identified = outlier_clipping(x, dx, method='weighted', clip=clip, side=side, verbose=True)
         detection_rates(outliers, identified, verbose=True)

         fig = plot_samples(x, outliers=outliers, identified=identified)
```

```
        fig.suptitle("Identified outliers (wsigma-clipping)", fontsize='large')

        mx = N.ma.masked_array(x, mask=identified)
        wm, dwm = sample_weighted_mean(mx, dx)
```

```
Outlier weighted-clipping: upper one-tailed, clip=3
  Iter #1: 1817 outliers (total: 1817)
  Iter #2: 85 outliers (total: 1902)
  Iter #3: 1 outliers (total: 1903)
Input outliers:      1959/40000 (4.90%)
Identified outliers: 1903 (TP: 1894, FP: 9, FN: 65)
Sensitivity TPR=1-FNR: 96.68%
Specificity TNR=1-FPR: 99.98%
Precision PPV:         99.53%
Accuracy:              99.81%
Matthews Correlation Coefficient: 0.980
```

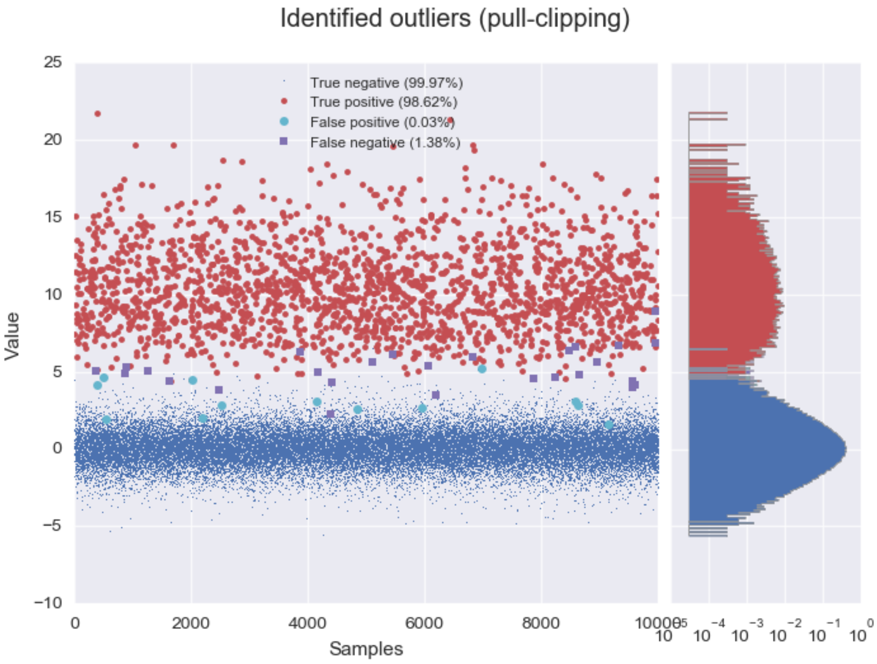


Pull-clipping In absence of a theoritically justified clipping value, we use here numerically-optimized clip=3.5 (see Appendix).

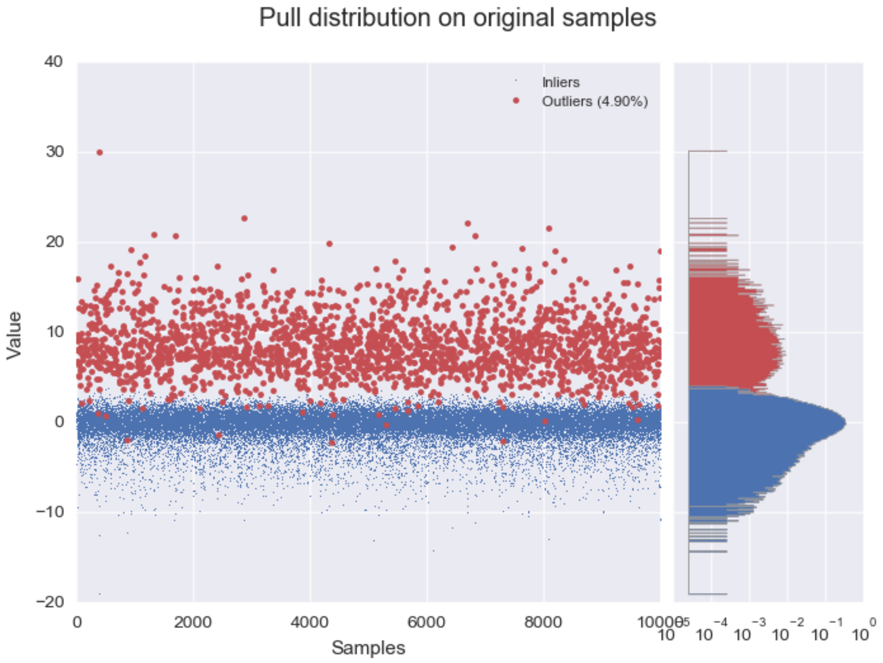
```
In [19]: clip = 3.5
         identified = outlier_clipping(x, dx, method='pull', clip=clip, side=side, verbose=True)
         detection_rates(outliers, identified, verbose=True)

         fig = plot_samples(x, outliers=outliers, identified=identified)
```

```
        fig.suptitle("Identified outliers (pull-clipping)", fontsize='large')

        mx = N.ma.masked_array(x, mask=identified)
        pm, dpm = sample_weighted_mean(mx, dx)
```

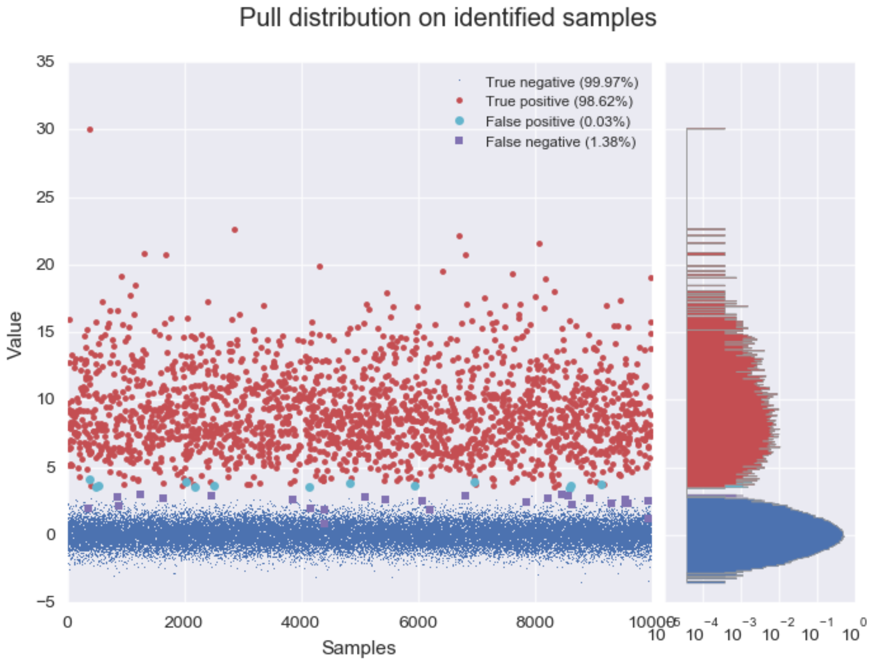
```
Outlier pull-clipping: upper one-tailed, clip=3.5
  Iter #1: 1813 outliers (total: 1813)
  Iter #2: 127 outliers (total: 1940)
  Iter #3: 5 outliers (total: 1945)
Input outliers:      1959/40000 (4.90%)
Identified outliers: 1945 (TP: 1932, FP: 13, FN: 27)
Sensitivity TPR=1-FNR: 98.62%
Specificity TNR=1-FPR: 99.97%
Precision PPV:         99.33%
Accuracy:              99.90%
Matthews Correlation Coefficient: 0.989
```



```
In [20]: p = pull(x, dx)
         fig = plot_samples(p, outliers=outliers)
         fig.suptitle("Pull distribution on original samples", fontsize='large');
```



```
In [21]: mp = pull(x, dx,
                   fixed_mean=pm.reshape(-1, 1), fixed_mean_error=dpm.reshape(-1, 1))
         fig = plot_samples(mp, outliers=outliers, identified=identified)

         fig.suptitle("Pull distribution on identified samples", fontsize='large');
```



5.3.4 Analysis

We compare now the results from the 3 promising estimators:

- `Median`: median on input sample
- `WSigmaC`: weighted mean on (weighted) σ_w-clipped sample
- `PullC`: weighted mean on pull-clipped sample.

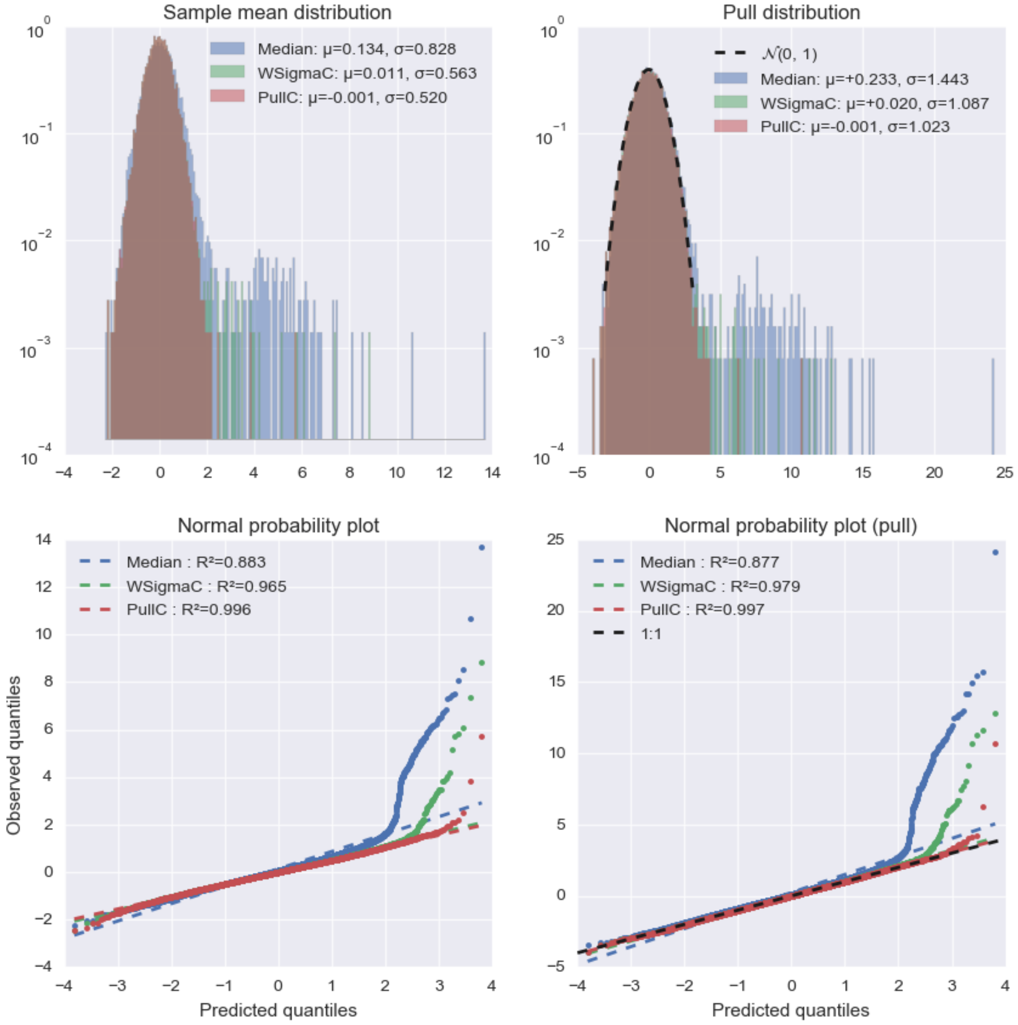
```
In [22]: fig, ((axh, axp), (axn, axq)) = P.subplots(2, 2, figsize=(10,10))

         _, bins = AS.freedman_bin_width(med, return_bins=True)
         axh.hist(med, bins=bins, normed=True, log=True, histtype='stepfilled', alpha=0.5,
                  label=u"Median: µ={:.3f}, ={:.3f}".format(med.mean(), med.std(ddof=1)))
         axh.hist(wm, bins=bins, normed=True, log=True, histtype='stepfilled', alpha=0.5,
                  label=u"WSigmaC: µ={:.3f}, ={:.3f}".format(wm.mean(), wm.std(ddof=1)))
         axh.hist(pm, bins=bins, normed=True, log=True, histtype='stepfilled', alpha=0.5,
                  label=u"PullC: µ={:.3f}, ={:.3f}".format(pm.mean(), pm.std(ddof=1)))
         axh.set(title="Sample mean distribution")
         axh.legend(loc='upper right', fontsize='small')

         print("\nMedian:")
         probplot(med, ax=axn, label='Median')
         plot_pull(med, dmed, fixed_mean=0, ax=axp, alpha=0.5, label='Median', log=True, normed=True)
         probplot(pull(med, dmed, fixed_mean=0), ax=axq, label='Median')
         print("\nWeighted sigma-clipping:")
         probplot(wm, ax=axn, label='WSigmaC')
         plot_pull(wm, dwm, fixed_mean=0, ax=axp, alpha=0.5, label='WSigmaC', log=True, normed=True)
```

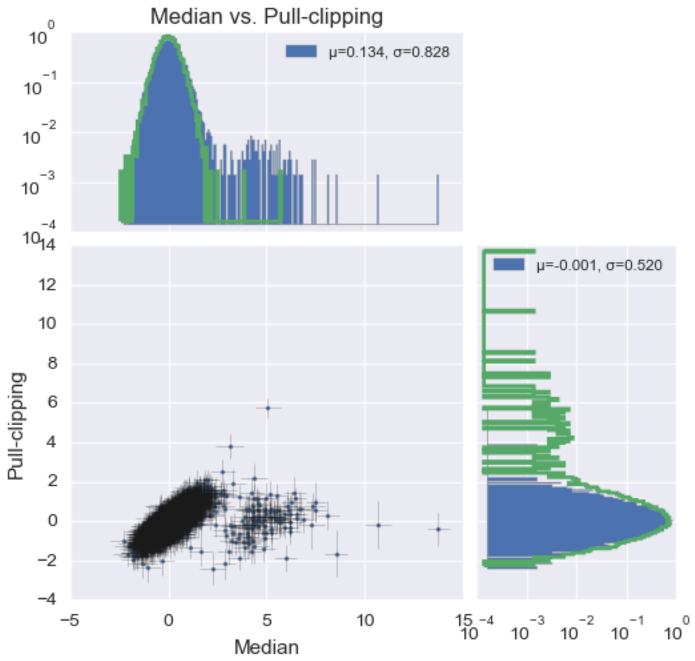
```
        probplot(pull(wm, dwm, fixed_mean=0), ax=axq, label='WSigmaC')
        print("\nPull-clipping:")
        probplot(pm, ax=axn, label='PullC')
        plot_pull(pm, dpm, fixed_mean=0, ax=axp, alpha=0.5, label='PullC', log=True, normed=True)
        probplot(pull(pm, dpm, fixed_mean=0), ax=axq, label='PullC')

        axn.set(title="Normal probability plot",
                xlabel="Predicted quantiles", ylabel="Observed quantiles")
        axn.legend(loc='upper left', fontsize='small')

        plot_rv(SS.norm, ax=axp, label='$\mathcal{N}$(0, 1)', c='k', lw=2, ls='--')
        axp.set_title("Pull distribution")
        axp.legend(loc='upper right', fontsize='small')

        axq.plot([-4, +4], [-4, +4], label='1:1', c='k', lw=2, ls='--')
        axq.set(title="Normal probability plot (pull)",
                xlabel="Predicted quantiles")
        axq.legend(loc='upper left', fontsize='small');
```

```
Median:
Probplot best-fit ax+b: a=0.731674, b=0.133618, R²=0.883152
Pull: mean=+0.233, std=1.443, normality p-value=0
Probplot best-fit ax+b: a=1.26595, b=0.233484, R²=0.876928

Weighted sigma-clipping:
Probplot best-fit ax+b: a=0.543685, b=0.0114969, R²=0.964701
Pull: mean=+0.020, std=1.087, normality p-value=0
Probplot best-fit ax+b: a=1.06442, b=0.0197317, R²=0.978903

Pull-clipping:
Probplot best-fit ax+b: a=0.51737, b=-0.00102278, R²=0.995551
Pull: mean=-0.001, std=1.023, normality p-value=6.97465e-72
Probplot best-fit ax+b: a=1.02024, b=-0.00133809, R²=0.997224
```

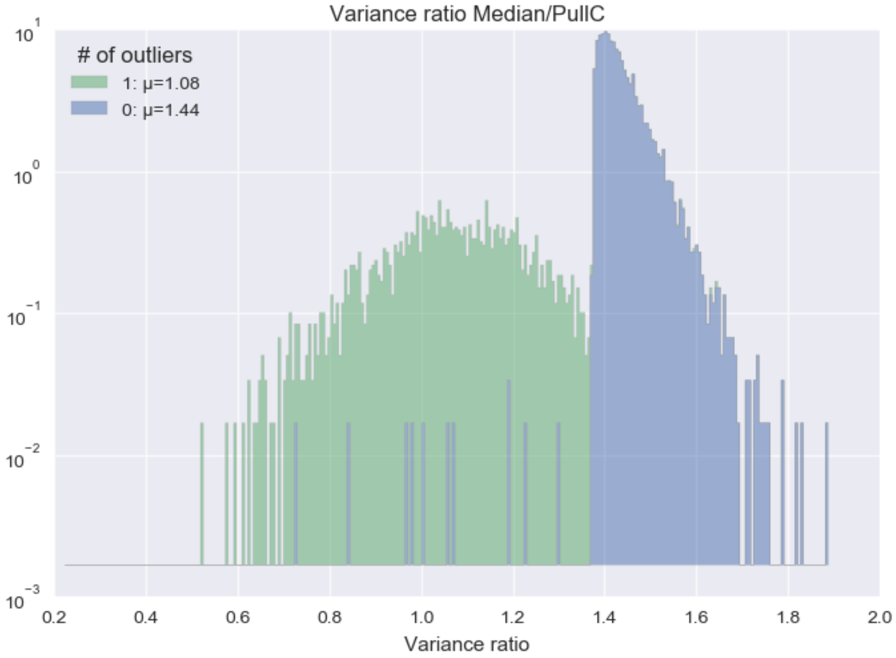
- The median distribution significantly deviates from the normal distribution starting at predicted quantile +2.1, i.e. for `1 - SS.norm.cdf(2.1)` = 1.8% of the samples. This corresponds to the fraction of samples with more than a single outlier, for which the median estimator breaks down. On the other hand, except for very few cases ($\lesssim 0.03\%$), the pull-clipped weighted mean is the least sensitive to outliers.
- All three pull distributions are well approximated by the $\mathcal{N}(0, 1)$ distribution (except for the outlying fraction), from which one can conclude that all three sample mean estimators are not significantly biased, and mean errors are well estimated.

```
In [23]: plot_comparison(x=med, dx=dmed, xlabel='Median',
                         y=pm, dy=dpm, ylabel='Pull-clipping');
```


Sample mean distribution
Median: μ=0.134, σ=0.828
WSigmaC: μ=0.011, σ=0.563
PullC: μ=-0.001, σ=0.520
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
−4
−2
0
2
4
6
8
10
12
14
Pull distribution
$\mathcal{N}(0, 1)$
Median: μ=+0.233, σ=1.443
WSigmaC: μ=+0.020, σ=1.087
PullC: μ=-0.001, σ=1.023
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
−5
0
5
10
15
20
25
Normal probability plot
Median : R²=0.883
WSigmaC : R²=0.965
PullC : R²=0.996
Observed quantiles
14
12
10
8
6
4
2
0
−2
−4
−4
−3
−2
−1
0
1
2
3
4
Predicted quantiles
Normal probability plot (pull)
Median : R²=0.877
WSigmaC : R²=0.979
PullC : R²=0.997
1:1
25
20
15
10
5
0
−5
−4
−3
−2
−1
0
1
2
3
4
Predicted quantiles



```
In [24]: rvar = (dmed / dpm.filled())**2  # Variance ratio (converted from masked array)
         _, bins = AS.freedman_bin_width(rvar, return_bins=True)

         fig, ax = P.subplots()
         ax.hist([ rvar[noutliers == i] for i in (0, 1) ], bins=bins,
                 normed=True, log=True, stacked=True, histtype='stepfilled', alpha=0.5,
                 label=[ u"{}: μ={:.2f}".format(i, rvar[noutliers == i].mean()) for i in (0, 1) ])
         ax.set(title='Variance ratio Median/PullC', xlabel='Variance ratio')
         ax.legend(title="# of outliers", fontsize='small', loc='upper left');
```



- In absence of outliers, the median is, as expected, significantly less efficient than the weighted mean, by 44% on average.
- Even in presence of a single outlier, the pull-clipped weighted mean still overperforms the median by 8% on average.

5.4 Conclusions

We tested three different approachs to compute the mean of a small-size ($n = 4$) sample in presence of a small fraction (5% probability) of outlying (10σ-level) values:

- the median of the sample,
- the inverse-variance weighted mean on (weighted) σ-clipped sample,
- the inverse-variance weighted mean on pull-clipped sample.

Among these 3 estimators, the *inverse-variance weighted mean on pull-clipped sample* appears to have the best performances:

- it is always statistically more efficient than the median, its variance being 30 to 40% smaller than the one of the median in absence of outliers (which is expected to be the vast majority of the cases, 81% in our experiment), and still 10% smaller in presence of a single outlier;
- it is less impacted by multiple outliers, given the very good performance of the pull-clipping process (as measured by a Matthews Correlation Coefficient of 99% in our experiment).

The results that were obtained here are probably slightly dependant of the details of the numerical experiment, notably the distribution adopted for the outliers (fraction, significance, impact on the measurement variance), however, the general conlusion would hold:

- the usage of the significantly sub-efficient median estimator is overdone when samples are only rarely affected by outliers,
- the pull-clipping is a performant procedure to detect outliers.

5.5 Appendix

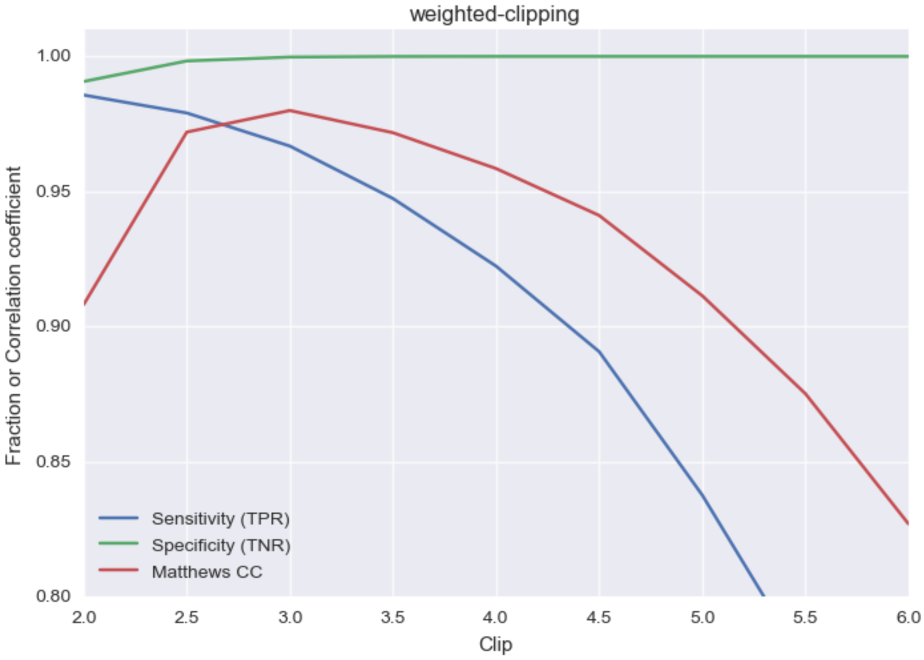
5.5.1 Clip study

We study now the dependency of the outlier detection efficiency (measured in terms of Matthews Correlation Coefficient) on the actual clipping value.

```
In [25]: def study_clip(x, dx, clips, method='pull', side=+1):
             """Detection rates as function of clip."""

             rates = N.array([detection_rates(outliers,
                                              outlier_clipping(x, dx, method=method, clip=clip, side=side),
                                              rates=('TPR', 'TNR', 'MCC'), verbose=False)
                              for clip in clips ])

             print(method)
             table = AT.Table([clips, rates[:, 0], rates[:, 1], rates[:, 2]],
                              names='Clip,TPR,TNR,Matthews CC'.split(','))
             print(table)

             fig = P.figure()
             ax = fig.add_subplot(1, 1, 1,
                                  xlabel='Clip',
                                  ylabel='Fraction or Correlation coefficient',
                                  title='{}-clipping'.format(method))
             ax.plot(clips, rates[:, 0], label='Sensitivity (TPR)')
             ax.plot(clips, rates[:, 1], label='Specificity (TNR)')
             ax.plot(clips, rates[:, 2], label='Matthews CC')
             ax.legend(loc='best', fontsize='small')
             ax.set_ylim(0.8, 1.01)

             return ax
```

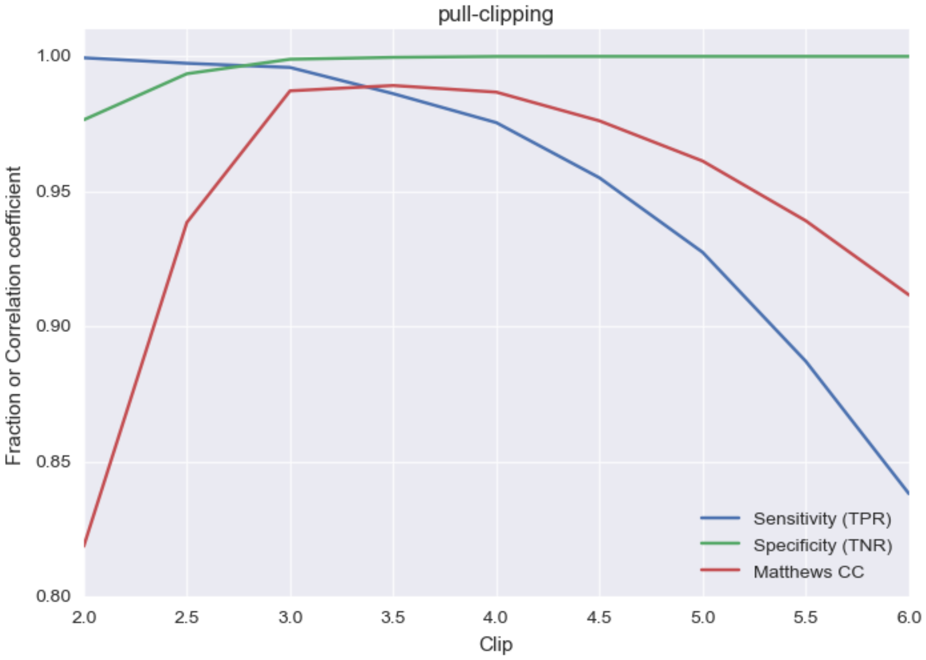
```
In [26]: clips = N.linspace(2, 6, 9)
         study_clip(x, dx, clips, method='weighted');
```

```
weighted
Clip      TPR            TNR        Matthews CC
---- -------------- -------------- --------------
 2.0 0.985706993364 0.990720538366 0.908180836409
 2.5 0.979070954569  0.99831760469 0.971994871145
 3.0 0.966819806023 0.999763413159 0.979980983818
 3.5  0.94742215416 0.999973712573 0.971765083323
 4.0 0.922409392547            1.0 0.958508424458
 4.5 0.890760592139            1.0  0.94115760729
 5.0 0.837672281776            1.0 0.911442753886
 5.5 0.774885145482            1.0 0.875217136842
 6.0 0.694742215416            1.0 0.827036863925
```



```
In [27]: study_clip(x, dx, clips, method='pull');
```

```
pull
Clip      TPR            TNR          Matthews CC
---- -------------- -------------- --------------
 2.0 0.999489535477 0.976525327936 0.818704993767
 2.5 0.997447677386 0.993585867879 0.938556242444
 3.0 0.995916283818 0.998948502931 0.987252831253
 3.5 0.986217457887 0.999658263453  0.98923548379
 4.0  0.97549770291 0.999973712573 0.986778548679
 4.5 0.955079122001            1.0 0.976153083677
 5.0 0.927514037774            1.0 0.961282828135
 5.5  0.88718734048            1.0 0.939182078632
 6.0 0.838182746299            1.0 0.911732305425
```


pull-clipping
Fraction or Correlation coefficient
1.00
0.95
0.90
0.85
0.80
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
Clip
Sensitivity (TPR)
Specificity (TNR)
Matthews CC