Variance stabilization with DDHFm

Guy Nason, Efthimios Motakis, Piotr Fryzlewicz
Statistics Group
Department of Mathematics
University of Bristol

October 19, 2005

Contents

1 Introduction

The *DDHFm* package is designed to perform data-driven Haar-Fisz (DDHF) variance stabilization. The basic DDHF method itself is described in [3, 4]. The modifications to DDHF to make it work successfully for microarray (or indeed similar kinds of replicate data) are described in [7].

The basic idea of the Haar-Fisz transform is very simple. First, a Haar wavelet transform is applied to the data. Then a Fisz transformation is applied to the wavelet coefficients. Then a standard inverse Haar transformation is applied to the Fisz-transformed wavelet coefficients to obtain a variance stabilized sequence. Informally, this kind of transformation works because the Fisz variance stabilizes (and moves towards Gaussian) each individual wavelet coefficient and then the multiscale nature of the inverse transform maintains the variance stabilization (since it is an orthogonal transform) by effectively enforcing the stabilization at every scale and location.

The data-driven HF transform adds in an extra step by trying to estimate the most effective power in the denominator of the Fisz step. Diagnostic information can be extracted from the DDHF transform which gives information as to the likely mean-variance relationship in the data.

More information on wavelets and the Haar wavelet transform can be found in [1] and more on the Fisz transform in [2]. The original paper on the HF transform for Poisson data can be found in [5].

2 Application of *DDHFm* to cDNA data

This section describes the application of *DDHFm* to a real set of cDNA data. The data can be obtained from the Stanford Microarray Database. The data arise from [6]: a study on mouse cDNA microarrays to investigate gene expression triggered by infection of bone marrow-derived macrophages with cytosol- and vacuole-localized Listeria monocytogenes (Lm). The experiment on each gene was replicated 4 times. Fluorescent labelled cDNAs were hybridized to compare the LLO^+ liposomes vs. LLO^- liposomes expression on the cytosol of mice. The data set numbers were 40430, 40571, 34905 and 34912.

To use DDHFm one needs to load the library and to access the cDNA data type:

```
> library("DDHFm")
> data(cdna)
```

The `cdna` matrix contains in formation on `nrow(cdna)` genes (rows) and `ncol(cdna)` replicates. The real value of DDHFm is when there are replicates and the more replicates the better.

At this stage it is worth plotting the "raw" data. This we can do with the following code:

```
> cdna.mn <- apply(cdna, 1, mean)
> cdna.sd <- apply(cdna, 1, sd)
> plot(cdna.mn, cdna.sd, xlab = "Replicates mean", ylab = "Replicates sd")
```

Figure 1 shows, for each gene, the standard deviation over replicates versus the mean. It is clear here that there is strong relationship between the variance and mean. Variance stabilization is a transform that attempts to make the variance *constant* as a function of the mean.

Now let us apply our DDHFm transform to the `cdna` data and construct a mean-variance plot similar to Figure 1.

```
> cdna.dd <- DDHFm(cdna)
> op <- par(fg = "gray90", tcl = -0.2, mgp = c(3, 0.5, 0))
> cdna.dd.mn <- apply(cdna.dd, 1, mean)
> cdna.dd.sd <- apply(cdna.dd, 1, sd)
> plot(cdna.dd.mn, cdna.dd.sd, xlab = "DDHFm rep mean", ylab = "DDHFm rep sd",
+     col = "gray90")
> library("lokern")
> cdna.dd.lo <- lokerns(x = cdna.dd.mn, y = cdna.dd.sd)
> lines(cdna.dd.lo$x.out, cdna.dd.lo$est, col = 2)
```

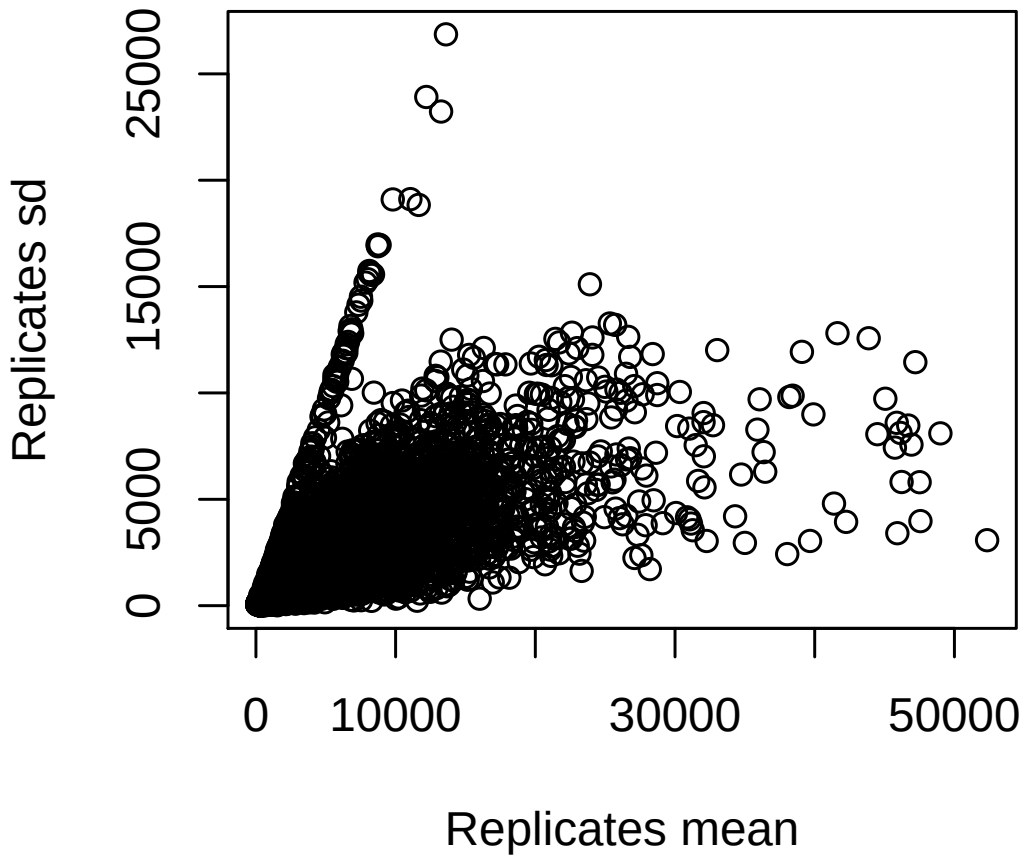

Figure 1: Replicate standard deviation versus the replicate mean. There is a strong mean-variance relationship

The DDHFm plot is shown in Figure 2. The red line is the best kernel smooth fit as determined by the `lokerns` function from the *lokern* package.

For an interesting comparison let us repeat this analysis but this time using `vsn` from the *vsn* package.

```
> library("vsn")
> cdna.vsn <- vsn(cdna)
> cdna.vsn.mn <- apply(cdna.vsn, 1, mean)
> cdna.vsn.sd <- apply(cdna.vsn, 1, sd)
> plot(cdna.vsn.mn, cdna.vsn.sd, xlab = "vsn rep mean", ylab = "vsn rep sd",
+     col = "gray90")
> cdna.vsn.lo <- lokerns(x = cdna.vsn.mn, y = cdna.vsn.sd)
> lines(cdna.vsn.lo$x.out, cdna.vsn.lo$est, col = 3)
```

The comparison between the DDHFm- and vsn-transformed data in Figures 2 and 3 is instructive. The green line for `vsn` is much less smooth than the comparable red one for `DDHFm`. Also, the median bandwidth for `vsn` is much less than that for `DDHFm` indicating that DDHFm has better stabilization. Recall further that these plots and kernel smooths are based on 42624 observations, a lot of data, and hence we have strong evidence in this case that DDHFm has stabilized better.

As a final comparison we compute the kernel smooth of the `vsn` transformed data but using the bandwidth from the `DDHFm` kernel smooth (the reason being that one might raise the objection that the `vsn` kernel smooth is only more variable because of the smaller bandwidth).

```
> cdna.vsn.lo.withsamebandwidthasdd <- lokerns(x = cdna.vsn.mn,
+     y = cdna.vsn.sd, inputb = TRUE, bandwidth = cdna.dd.lo$bandwidth)
> plot(seq(from = 0, to = 1, length = 300), cdna.vsn.lo$est, ylim = c(0,
+     1.1), xlab = "x", ylab = "VSN transformed scale", type = "n")
> lines(seq(from = 0, to = 1, length = 300), cdna.vsn.lo$est, col = 3)
> lines(seq(from = 0, to = 1, length = 300), cdna.vsn.lo.withsamebandwidthasdd$est,
+     col = 4)
> lines(seq(from = 0, to = 1, length = 300), cdna.dd.lo$est * 0.4988,
+     col = 2)
```

As Figure 4 shows even when the `vsn` data are smoothed using the identical bandwidth to the DDHFm kernel smooth it (the blue line) is still more variable than the DDHFm kernel smooth (red line). On the other hand one could look solely at the median smoothing bandwidths for the kernel smoothers on the `vsn` and DDHFm transformed data (0.4663 and 1.8) and this is evidence itself that the `vsn` result here is less stabilized than DDHFm.

It is important to realize that the improvement due to DDHFm is in an *overall* sense and much of the improvement is due to DDHFm incorporating replicate information in

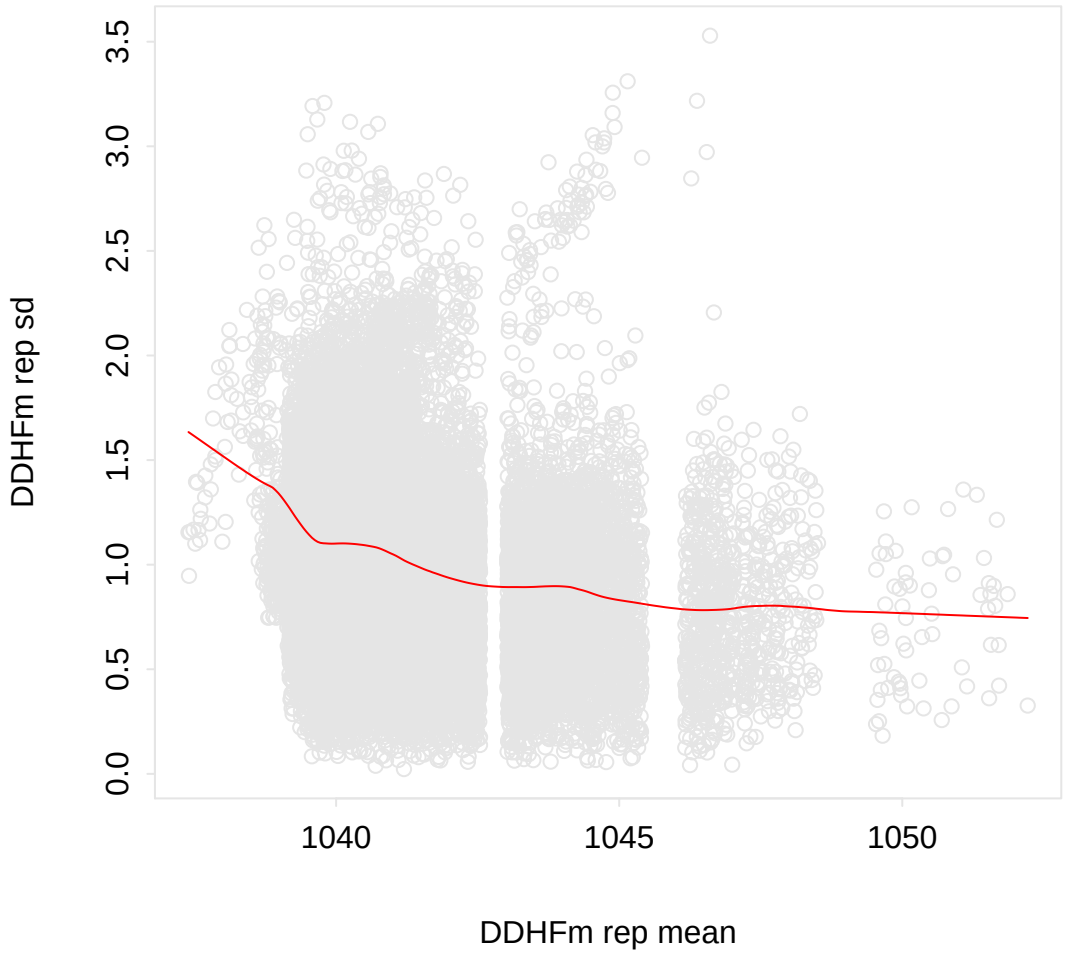

Figure 2: Replicate standard deviation versus the replicate mean for the DDHfm transformed data. Red line is best `lokerns` kernel smoothing fit which is not far from horizontal indicating good stabilization. The median bandwidth is 1.8.

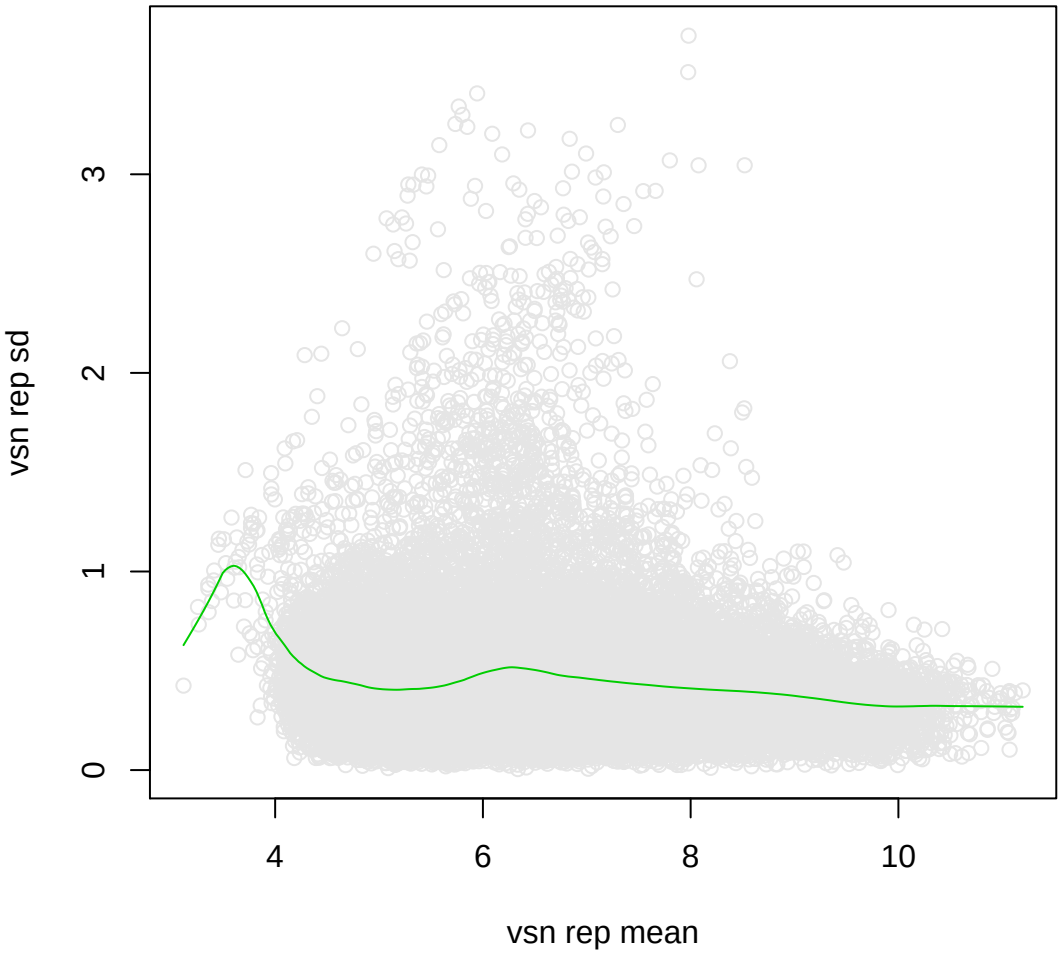

Figure 3: Replicate standard deviation versus the replicate mean for the vsn transformed data. The green line is best `lokerns` kernel smooth fit. The median bandwidth was 0.4663. The fitted line has more dips and peaks than the comparable on DDHFm fit.

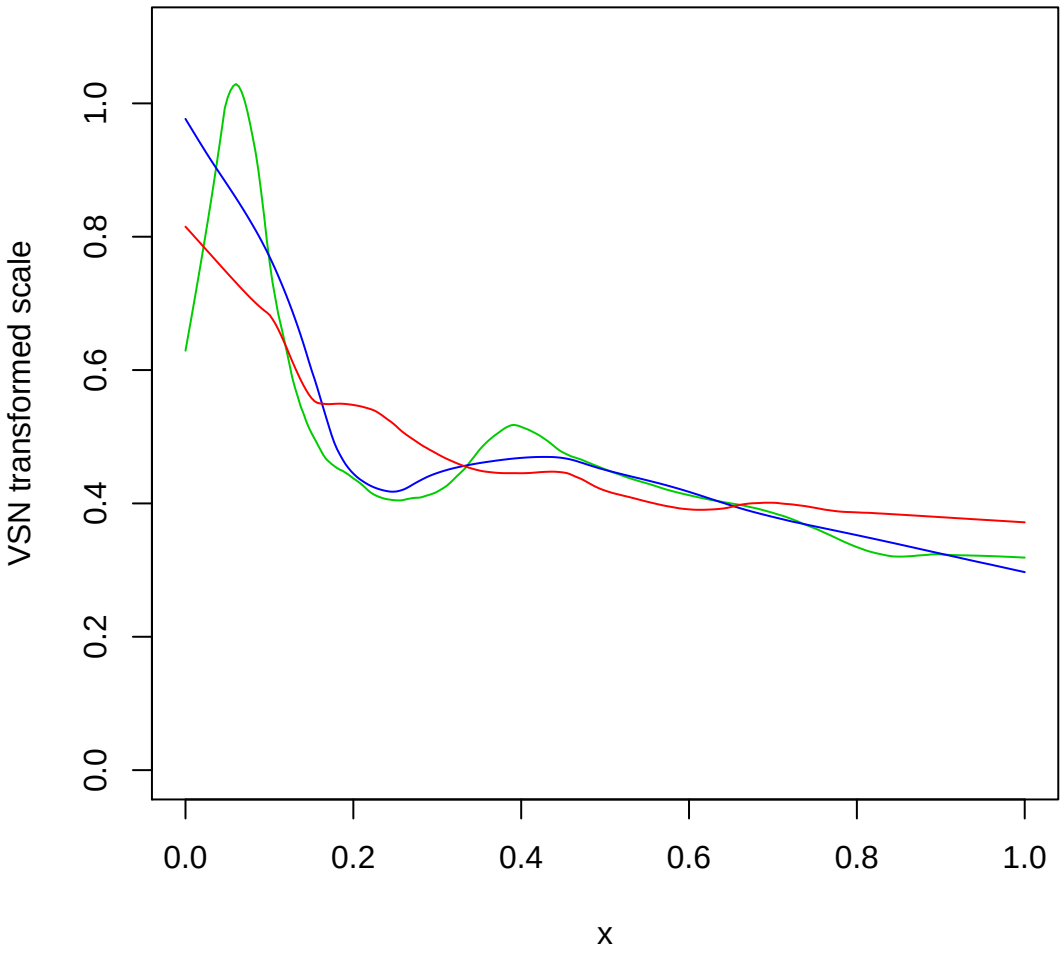

Figure 4: Kernel smooths: (green) kernel smooth of `vsn`-transformed data from Figure 3 with median bandwidth of 0.4663; (blue) kernel smooth of `vsn`-transformed data using bandwidth array from the DDHFm kernel smooth (median bandwidth 1.8); (red) kernel smooth of `DDHFm`-transformed data with median bandwidth of 1.8 but rescaled to have same vertical scale as (green) `vsn` kernel smooth.

a systematic way. The `vsn` variance stabilization works independently on each variable and does not formally take account of replication. As a result on a per-variable basis `vsn` *does* perform better. However, as technologies improve replication and increasing numbers of replicates will mean that methods such as DDHFm which explicitly take account of replicates will become even more valuable.

3 Application of the basic DDHF algorithm

The previous section demonstrated the application of the DDHFm method to cDNA data. The `DDHFm` function relies on the `ddhft.np.2` function to apply the actual data-driven Haar-Fisz algorithm. Here follows an example of using the DDHF algorithm directly.

3.1 Poisson data

For this example, we will first create a Poisson intensity vector and plot it in Figure 5. First, let us create a function that creates a Poisson intensity vector (of any size, mod 4).

```
> makepiv <- function(nbit = 4) {
+     p1 <- rep(3, nbit)
+     p2 <- rep(5, nbit)
+     p3 <- rep(1, nbit)
+     p4 <- rep(10, nbit)
+     xx <- seq(from = 1, to = 10, length = nbit * 4)/3
+     p5 <- xx^2
+     c(p1, p2, p3, p4, p5)
+ }
```

We'll make a Poisson intensity vector of length 256 and this is plotted in Figure 5.

```
> piv <- makepiv(nbit = 32)
> l <- length(piv)
> ts.plot(piv, xlab = "x", ylab = "Intensity vector")
```

Now let us sample a Poisson sequence with this intensity vector and plot this in Figure 6.

```
> pseq <- rpois(l, lambda = piv)
> plot(1:l, pseq, xlab = "x", ylab = "Poisson sample from intensity vector")
> lines(1:l, piv, lty = 2)
```

Now we will apply the Haar-Fisz transform.

```
> pseq.ddhf <- ddhft.np.2(pseq)
> plot(pseq.ddhf$mu, pseq.ddhf$sigma2, xlab = "Estimated mu", ylab = "Estimated sd")
> lines(pseq.ddhf$mu, pseq.ddhf$sigma)
> lines(pseq.ddhf$mu, sqrt(pseq.ddhf$mu), lty = 2)
```

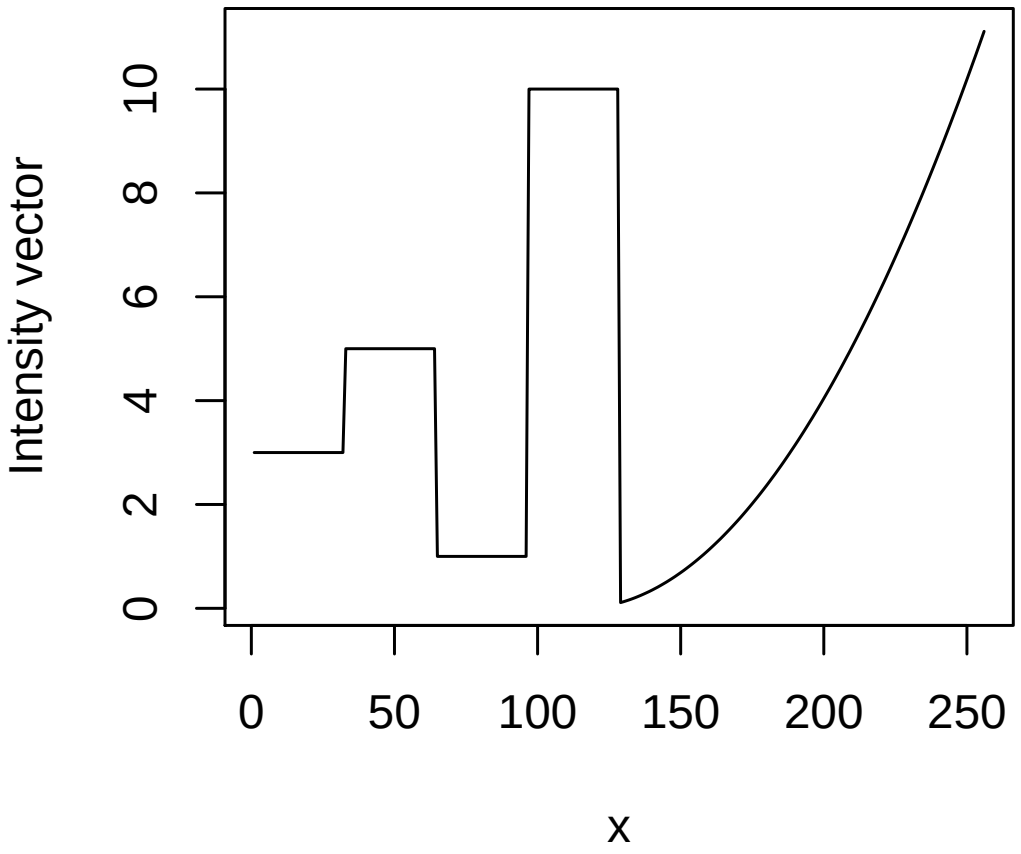

Figure 5: Contrived Poisson intensity vector.

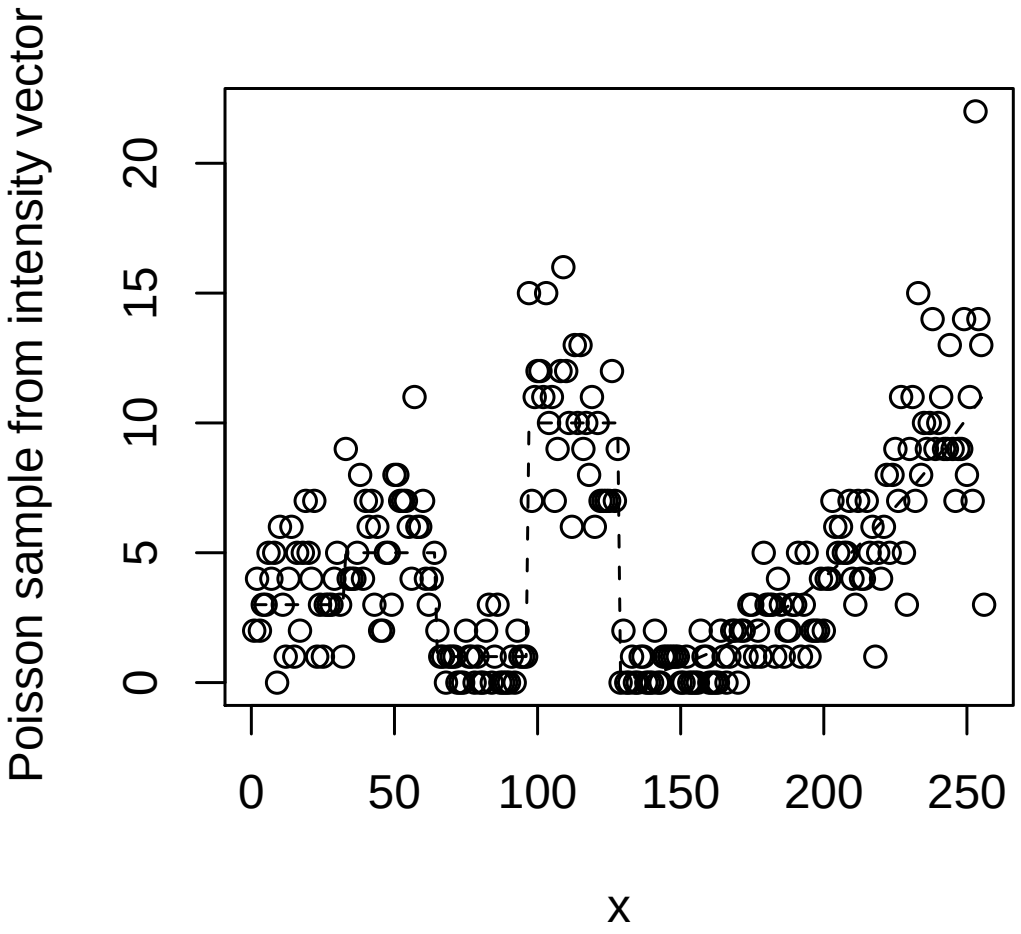

Figure 6: Poisson sample realization from intensity vector.

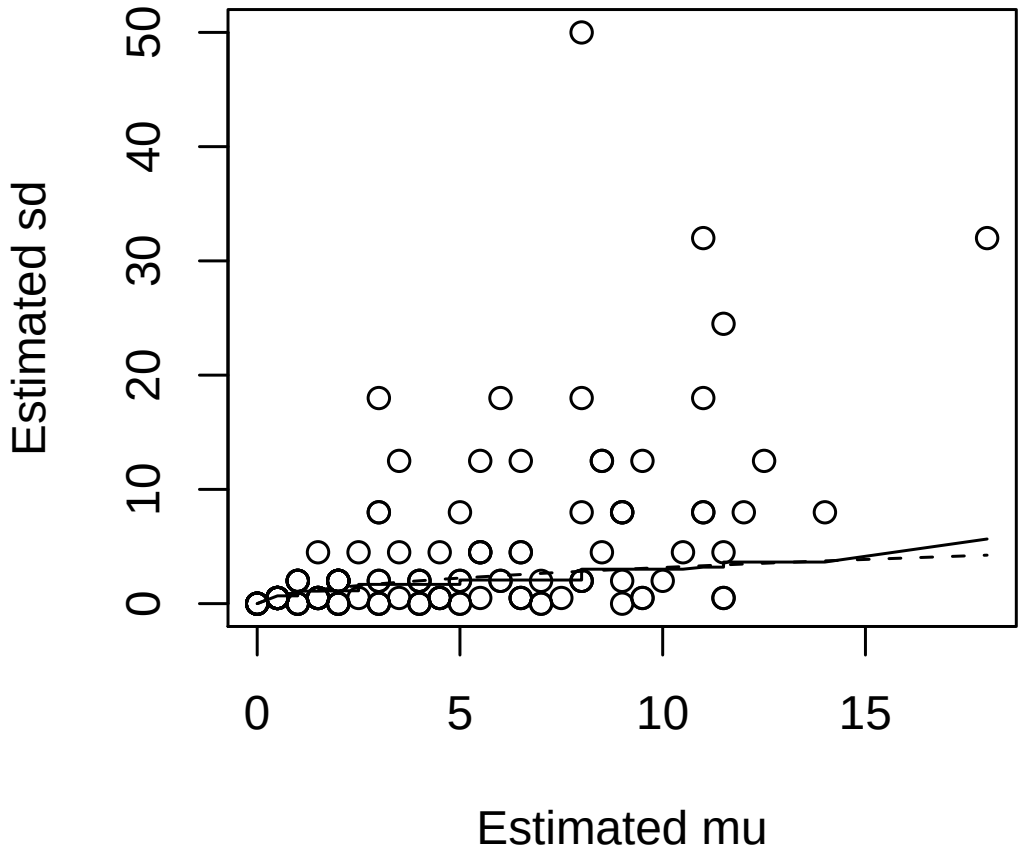

Figure 7: Computed mean-variance relationship (dots), estimated mean-variance relationship (isotone regression, blocky line), sqrt root function for comparison purposes (dashed line).

Figure 7 shows the results of the DDHF algorithm. It shows the computed mean-variance relationship from the data by small circles. The values indicated by the circles are the mother and father Haar wavelet coefficients at the finest scale. The blocky solid line indicates the best isotonic regression fit (which finds the best monotonically increasing mean-variance relationship). Figure 7 also plots the ordinary square-root function in a dashed line. Notice how close the isotonic regression gets to the square root function. This is because in this case for Poisson random variable the mean equals the variance and so we have that the standard deviation is proportional to (actual equal to in this case) the square root of the mean.

For estimation let us apply some light smoothing to the DDHF-transformed data.

```
> pseq2.ddhf <- pseq.ddhf
> library("wavethresh")
> hftwd <- wd(pseq.ddhf$hft, filter.number = 1, family = "DaubExPhase")
> madmad <- function(x) mad(x)^2
> hftwdT <- threshold(hftwd, policy = "universal", levels = hftwd$nlevels -
+     1, dev = madmad, return.thresh = TRUE)
> hftwd.thresh <- threshold(hftwd, policy = "manual", value = hftwdT)
> hftwr <- wr(hftwd.thresh)
> pseq2.ddhf$hft <- hftwr
> plot(1:l, pseq.ddhf$hft, xlab = "x", ylab = "Haar wavelet fit to DDHF data")
> lines(1:l, hftwr)
```

In the transformed domain Figure 8 shows the transformed data and a universal threshold Haar wavelet shrinkage fit to it.

We then transform the Haar wavelet shrunk estimate back into the original domain using the `ddhft.np.inv` function and plot the results in Figure 9.

```
> pest2 <- ddhft.np.inv(pseq2.ddhf)
> plot(1:l, pseq, xlab = "x", ylab = "Poisson data")
> lines(1:l, pest2)
> lines(1:l, piv, col = 2, lty = 2)
> lines(1:l, pss <- smooth.spline(1:l, y = pseq)$y, col = 3)
> hfssq <- sum((pest2 - piv)^2)
> ssssq <- sum((pss - piv)^2)
> title(sub = paste("SSQ: DDHF=", round(hfssq, 0), " SS=", round(ssssq,
+     0)))
```

Note in Figure 9 that the DDHF-transformed estimate is closer to the truth and is less variable than that computed directly on the Poisson data (or should be). Possibly a better comparison would be a wavelet shrunk estimate instead of a smoothing spline estimate (but this is a question of the smoothing employed and not of the transform).

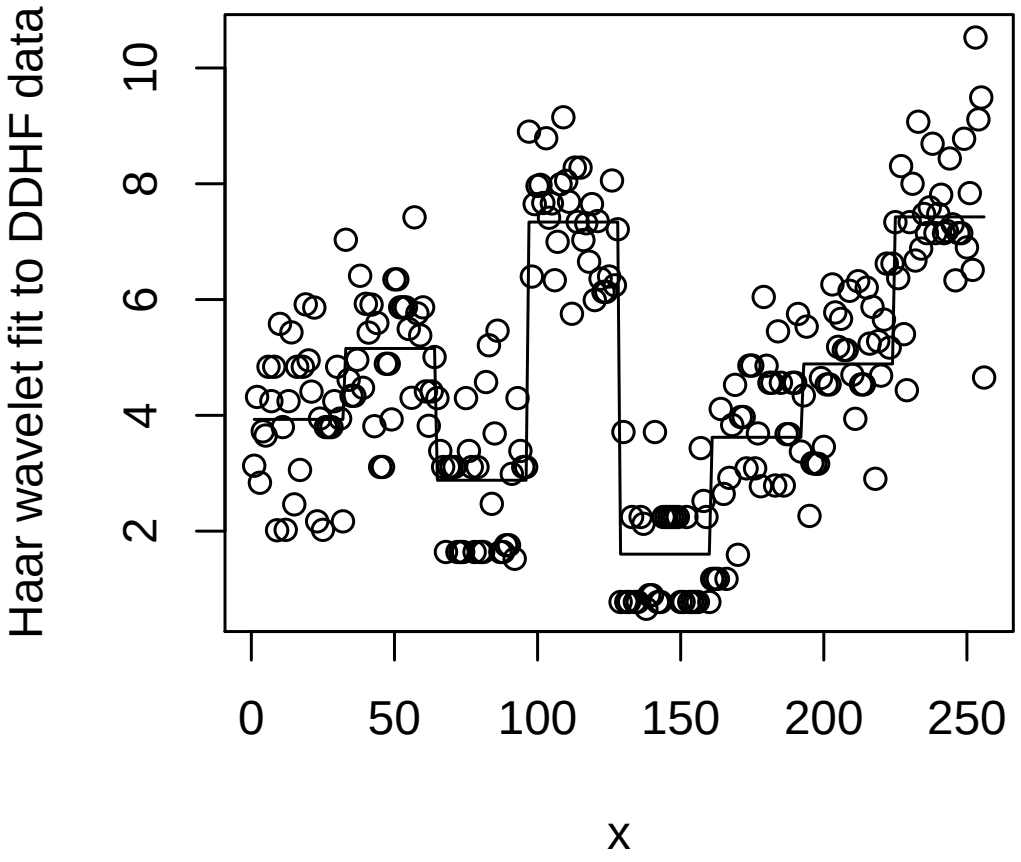

Figure 8: DDHF-transformed data (points) and light smoothed Haar wavelet fit to this data (line).

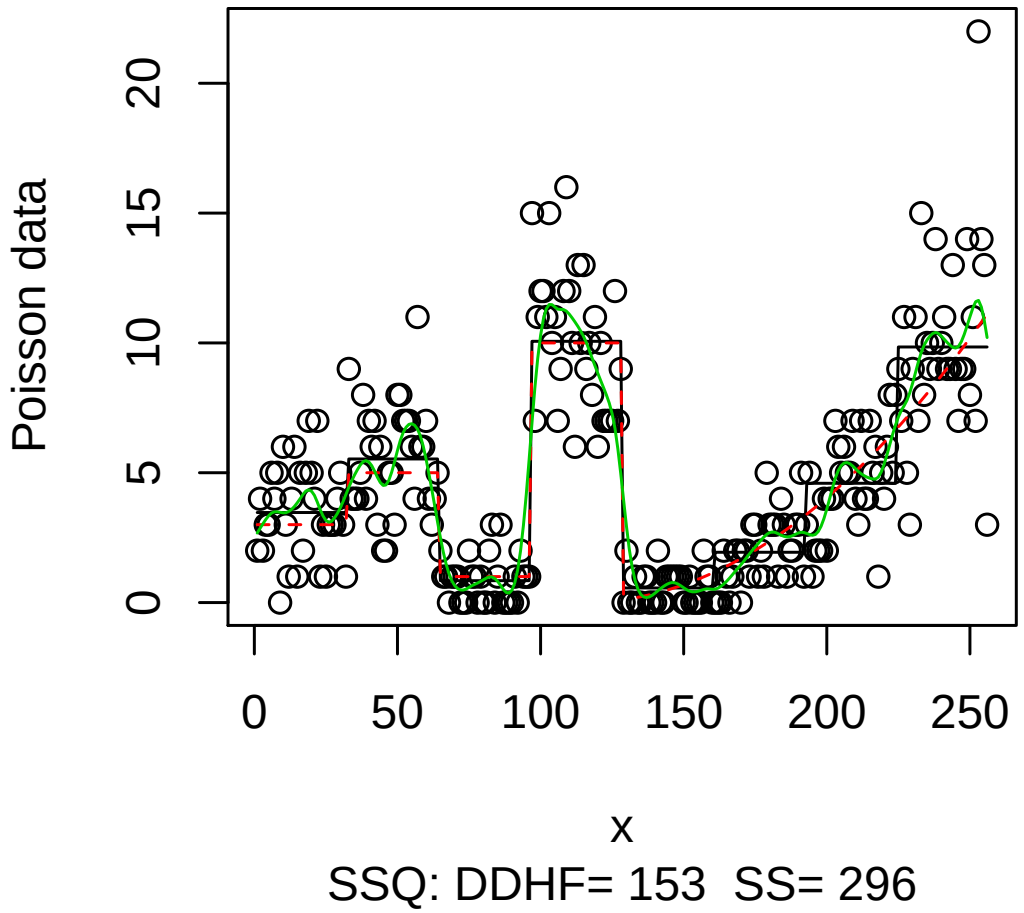

Figure 9: Original Poisson distributed data (points) with true intensity function (red line). The black line is the intensity estimate obtained through DDHF transformation (transform-smooth-invert) and the green line is the direct smoothing spline estimate using the Poisson data as input. (SSQ is error measure).

References

[1] Burrus, C.S., Gopinath, R.A. and Guo, H. (1998) *Introduction to Wavelets and Wavelet Transforms: a Primer.* Prentice-Hall: Upper Saddle River, NJ. 2

[2] Fisz, M. (1955) The limiting distribution of a functon of two independent random variables and its statistical application. *Colloquium Mathematicum*, **3**, 138–146. 2

[3] Fryzlewicz, P. and Delouille, V. (2005) A data-driven Haar-Fisz transform for multiscale variance stabilization. Proceedings of the 2005 IEEE Workshop on Statistical Signal Processing. 1

[4] Fryzlewicz, P., Delouille, V. and Nason, G.P. (2005) A data-driven Haar-Fisz transform for multiscale variance stabilization. *Technical Report* 05:06, Statistics Group, Department of Mathematics, University of Bristol, UK 1

[5] Fryzlewicz, P. and Nason, G.P. (2004) A Haar-Fisz algorithm for Poisson intensity estimation. *Journal of Computational and Graphical Statistics*, **13**, 621–638. 2

[6] McCaffrey, R.L., Fawcett, P., O'Riordan, M., Lee, K., Havell, E.A., Brown, P.O. and Portnoy, D.A. (2004) A specific gene expression program triggered by Gram-positive bacteria in the cytocol. *Proc. Nat. Acad. Sci.*, **101**, 11386–11391. 2

[7] Motakis, E.S., Nason, G.P., Fryzlewicz, P. and Rutter, G.A. (2005) Variance stabilization and normalization for one-color microarray data using a data-driven multiscale approach. *Technical Report* 05:16, Statistics Group, Department of Mathematics, University of Bristol, UK 1