Description of rHVDM: an R/Bioconductor package implementing Hidden Variable Dynamic Modelling

Martino Barenco

October 13, 2014

Contents

Background

The rHVDM package is an improved version of Hidden Variable Dynamic Modelling (HVDM) [1][1] implemented in R for speed and convenience. HVDM is a mathematical

[1] `http://genomebiology.com/2006/7/3/R25`

technique which predicts a given transcription factor activity and then uses this information to predict its targets. To achieve that, HVDM uses time course expression data inconjunction with a dynamic model. HVDM takes advantage of prior biological knowledge to create a training set of genes, the behaviour of which can be used to derive the activity profile of the controlling transcription factor. This activity parameter -the hidden variable- can then be used to identify other targets of the same factor ranked in order of likelihood.

The new R implementation incorporates two significant improvements over the original C version. First, we have introduced a more efficient optimisation technique (Levenberg-Marquardt) which significantly improves on the Nelder-Mead method applied before. Second, the Hessian matrix created as a by-product of the new method can be used to assess the robustness of the outcome. This results in a faster performance compared to the original implementation where confidence intervals were estimated via a comparatively slow Markov Chain Monte Carlo approach. rHVDM can now analyse microarray time series data from a thousand genes in under 5 minutes to accurately predict targets of a transcription factor.

The most important section of the present document is section 2, which contains a step-by-step sample session; the data for this session are supplied with the package. Going through this session should take about an hour and we strongly recommend doing this before trying *rHVDM* with your own data. That section is preceded by the data requirements description and a short introduction on the mathematical principles behind the package. It is followed by more technical aspects of *rHVDM* (Accessing information in *rHVDM* objects, confidence intervals computation, implementation changes).

Type of data required

rHVDM requires the following input information:

- Expression time course microarray data, consisting of at least five time points. Note that the technique can cope with irregular sampling.
- Some prior biological knowledge about the transcription factor under review: at least three genes should be known to be targets of that transcription factor and presumed to be targets of that transcription factor *only*. These genes constitute the *training set*.
- The transcript degradation rate of one the known targets (measured in an independent experiment (for example by Q-RTPCR).
- The measurement error for each expression value should be known. Note that this is the technical error, not the biological error. See below and section 2.1 for more details about this.

These requirements are the bare minimum for *rHVDM* to produce meaningful results. More time points and replication improve results but are not essential. Little advantage is gained by increasing the number of training set genes beyond five. No advantage is gained by measuring more than one degradation rate. The data with which this technique was originally developed (and used in the example given below) consists of a time course run in triplicates comprising each seven time points, sampled regularly every two hours. We used more than three known targets (five). Surprisingly, biological replicates are not required but do help achieving more robust results.

The experimental design should be such that the activity of the transcription factor under review changes in time. A flat activity (following a knock down of that transcription factor for example) would not carry much information. The time range covered by the time course should be loosely commensurate with the half lives of the transcripts of the target genes. In our example, the experiment covers twelve hours and the half life of a typical transcript is in the range of hours. Conversely, an experiment covering a week with a snapshot every twenty-four hours is unlikely to be informative.

Crucially, something should be known about the measurement errors (*ie* technical rather than biological error), in particular their variance. The idea behind this is that the fitting should be more lenient with those expression values that are more noisy and, more generally, that the uncertainty attached to the data should somehow "trickle down" to the parameter estimation. Broadly speaking, the lower the expression level, the higher the associated error is. This is important in HVDM as the robustness of one of those parameters is critical in indicating whether a particular transcript is a putative target of the transcription factor under examination. Starting from version 1.5 of the package, this measurement error can be estimated from the data (see subsection 2.1 for more details).

1 HVDM uncovered

Before giving a sample session (in the next section), we briefly present in the present section the mathematical basis for the algorithm.

1.1 The model

The technique supposes that the expression of a given gene j is governed by the following ordinary differential expression:

$$\frac{dX_j(t)}{dt} = B_j + S_j f(t) - D_j X_j(t) \tag{1}$$

where $X_j(t)$ denotes the concentration of gene j at time t. The left-hand side of this equation denotes the rate of change of that concentration while the right-hand side gathers all the terms having an influence on the rate of change. The first term, B_j is a *basal* rate of change. The second term $S_j f(t)$ is the rate of change that is dependent on the

transcription factor. The constant is S_j the sensitivity of gene j to the transcription factor activity (and hence is specific to that gene), while $f(t)$ is the transcription factor activity at time t and may apply to other genes, provided they are targets of the transcription factor. The last term is the degradation term, we suppose it is proportional to the transcript concentration. Hence, like B_j and S_j, the degradation rate D_j is a constant parameter. In a microarray time course experiment, we have measurements for $X_j(t)$, or rather we have approximate values $\hat{X}_j(t)$ because expression values are hampered by measurement errors:

$$\hat{X}_j(t_i) = X_j(t_i) + \epsilon_{j,i} \tag{2}$$

where $\epsilon_{j,i}$ is a random variable denoting the *measurement error*. We suppose $\epsilon_{j,i}$ to be Gaussian with expectancy zero and variance $\sigma^2_{j,i}$. All the other quantities in equation (1) remain unknown. We are in particular interested in $f(t)$, the transcription factor activity which is the "hidden variable". To find the transcription factor activity, previous knowledge about the biological system under review is used, *ie* known targets of the transcription factor and the degradation rate of the mRNA of one of those targets (the latter should be measured independently, using for example Q-RTPCR). Hence in an initial training step, optimal values for the kinetic parameters B_j, S_j, D_j for the training genes and the activity profile $f(t)$ are found. In the second (screening) step, the model is fitted to other transcripts' expression data, with $f(t)$ known. Those genes transcripts for which the model fits the data well and whose sensitivity S_j is significantly larger than zero are considered putative targets of the transcription factor under review.

1.2 More details about the fitting

The transcription factor activity variable $f(t)$ is in reality a continuous function. However, any time course experiment consists only of a limited amount of measurements at discrete time points, which we will denote henceforth as $\hat{\mathbf{X}}_\mathbf{j}$, where $\hat{\mathbf{X}}_\mathbf{j} \equiv \{\hat{X}_j(t_0), \hat{X}_j(t_1), \ldots, \hat{X}_j(t_n))\}$. Hence, we aim to estimate $f(t)$ at the same time points: $\mathbf{f} \equiv \{f(t_0), f(t_1), \ldots, f(t_n))\}$.

Similarly to estimate the left-hand side of (1), the rate of change of the transcript concentration, we use a differential operator A, which is a square matrix with the same dimensions as there are time points in the time course. The entries of this matrix are calculated using polynomial interpolation. These entries depend only on the timing of the measurement, not on the expression value. More details can be found in the supplementary material of [1]. Hence we have an approximate (and discrete) version of (1):

$$A\mathbf{X}_\mathbf{j} = B_j\mathbf{1} + S_j\mathbf{f} - D_j\mathbf{X}_\mathbf{j} \tag{3}$$

whose formal solution[2] for $\mathbf{X}_\mathbf{j}$ is given by

$$\mathbf{X}_\mathbf{j} = (A + D_j\mathbf{I})^{-1}(B_j\mathbf{1} + S_j\mathbf{f})$$

[2]In practice, this system of linear equations is solved using LU substitution.

where $\mathbf{I}$ is the identity matrix and $\mathbf{1}$ a vector whose every single entry is 1.

Fitting the model amounts to finding the parameter values for which $\mathbf{X_j}$ (modelled expression values) are closest to $\hat{\mathbf{X}}_\mathbf{j}$ (observed expression values). As a measure of "closeness" we use the following expression, which we will call the *model score*:

$$M(\mathbf{p}) = \sum_{genes,timepoints,replicates} \left(\frac{\hat{X}_j(t_i) - X_j(t_i)}{\sigma_{j,i}} \right)^2 \tag{4}$$

In the equation above, $\sigma_{j,i}$ is the standard deviation of the measurement error in (2) and $\mathbf{p}$ is the set of parameters to be fitted. The contents of this set depends on the context. In the training step, $\mathbf{p}$ comprises the kinetic parameters of the training genes as well as the transcription activity profile, while in the screening step, only the kinetic parameters of the gene under review are fitted. Fitting the model amounts to finding the values of $\mathbf{p}$ that minimize $M(\mathbf{p})$ above. It is worth noting, that in the case where there are several replicates in the experiment, a different transcription activity profile is used for each replicate, whilst the kinetic parameters for individual genes are supposed to be the same across replicates.

The fitting algorithm used in the R implementation of HVDM is Levenberg-Marquardt (LM), which is gradient-based and applied using the excellent *minpack.lm* package. A useful by-product of LM is the hessian matrix H. H is an approximation of the partial second derivatives of the model score function (4) with respect to individual elements of $\mathbf{p}$, the vector of parameters. This matrix is instrumental in leading the optimisation but also has interesting properties when it comes to evaluating the robustness of the fit. Inverting H[3] gives an approximation of the covariance matrix of the fitted parameters, which we use as a measure of robustness. For example, the diagonal of H^{-1} contains the variance of the individual parameters. It is also possible to derive, from the Hessian matrix, useful information about the quality of the fit (see next section).

2 A sample session

The *rHVDM* package comprises five functions, `HVDMcheck`, `HVDMreport`, `training`, `fitgene` and `screening`. The first two are *diagnostic* commands and the last three perform various types of fitting. In this section we demonstrate these commands using the example presented in [1]. The relevant data for this sample session is attached with the package. *rHVDM* requires three other packages: *affy*[4] [2] (for data handling), *R2HTML* [3] (for report generation) and *minpack.lm* (for the optimisation step). Once these packages and *rHVDM* have been installed, *rHVDM* can be launched in the R session using:

```
> library(rHVDM) ##the three other packages also get loaded
```

[3] If H is non-singular, which is not guaranteed.

[4] Normally, *Biobase* should be enough but we specify *affy* to ensure Windows compatibility.

To load the data that will be used in this sample session[5]:

```
> data(HVDMexample)
```

This loads the three R objects: `fiveGyMAS5`, `p53traingenes` and `genestoscreen`. `fiveGyMAS5` contains the data set to be used, `p53traingenes` and `genestoscreen` are vectors containing gene identifiers. The data is of human T-cells (MOLT4 line) that were subjected to a 5 Gy dose of γ-irradiation. This causes DNA double-strand breaks which elicits a complex transcriptional response in the cell. Messenger RNA was collected every two hours and up to twelve hours after irradiation, as well as just before irradiation (which is the zero hours, or control, time point) and affymetrix HGU133A arrays were hybridized using standard procedures. Thus there are seven time points $\{0, 2, 4, 6, 8, 10, 12\}$ and the experiment was run in triplicates. Probe level data were summarized using MAS5.0 and rescaled [4][6]. It is these data that are contained in the `fiveGyMAS5` object. In this sample session, we will apply *rHVDM* to identify targets of the p53 tumor suppressor.

2.1 Data requirements

The time course data used should:

- be contained in an `ExpressionSet` object (*Biobase* package). From version 1.1 onwards, *rHVDM* won't accept `exprSet` objects. To convert the old structure into the new use the `as` command:

  ```
  ><ExpressionSet-object> <- as(<exprSet-object>,"ExpressionSet")
  ```

- The expression values should be summarized (*ie*, probe level data will not work).

- The expression values should be in "raw" form (*ie* not log-transformed).

In addition to expression values, estimates for the standard deviation of the measurement error *have to be supplied* (in the `se.exprs` slot of the expression set). A "quick and dirty" way to create some of those would be the following command (It creates a new ExpressionSet "on the fly"):

```
> anothereset<-assayDataElementReplace(fiveGyMAS5,'se.exprs',5 + exprs(fiveGyMAS5)*0.
```

This supposes the noise to be 10 percent of the signal intensity, plus an additive component. Recall that this is a "quick and dirty" method.

From version 1.5 onwards, rHVDM implements, with some minor modifications, the method described in the Genome Biology paper [1, 5, 6, 7].

[5] Under Linux, before launching the `data` command one needs to load the datasets library using `library(datasets)`.

[6] `http://www.biomedcentral.com/1471-2105/7/251`

```
> anothereset<-estimerrors(eset=fiveGyMAS5,plattid="affy_HGU133A")
```

The `plattid` parameter gives the plattform. To check what plattforms are available, type the `estimerrors` command with no arguments.

Before carrying on with this session, we remove the spurious expression set:

```
> rm(anothereset)
```

as `fiveGyMAS5` contains everything we need already. The pheno data of the expression set should contain specific fields. To access this phenodata, use the *Biobase* command:

```
> pData(fiveGyMAS5)
```

	replicate	time	experiment
cARP3-6hrs.CEL	3	6	5Gy
cARP3-2hrs.CEL	3	2	5Gy
cARP3-0hrs.CEL	3	0	5Gy
cARP2-0hrs.CEL	2	0	5Gy
cARP2-12hrs.CEL	2	12	5Gy
cARP1-6hrs.CEL	1	6	5Gy
cARP1-12hrs.CEL	1	12	5Gy
cARP1-2hrs.CEL	1	2	5Gy
cARP3-8hrs.CEL	3	8	5Gy
cARP2-8hrs.CEL	2	8	5Gy
cARP3-12hrs.CEL	3	12	5Gy
cARP1-0hrs.CEL	1	0	5Gy
cARP2-4hrs.CEL	2	4	5Gy
cARP1-8hrs.CEL	1	8	5Gy
cARP2-10hrs.CEL	2	10	5Gy
cARP2-2hrs.CEL	2	2	5Gy
cARP1-4hrs.CEL	1	4	5Gy
cARP2-6hrs.CEL	2	6	5Gy
cARP3-4hrs.CEL	3	4	5Gy
cARP1-10hrs.CEL	1	10	5Gy
cARP3-10hrs.CEL	3	10	5Gy

The pheno data should contain at least three fields: `replicate` (even if there is only one replicate), `time` and `experiment`. Of those three, only the `time` field has to contain numerical values (in the two other fields, entries can have a numeric or character format). In addition, there should be a zero time point per (experiment/replicate), there should be no duplicate time values per (experiment/replicate) and no negative time. The row names in the pheno data should be consistent with the column names of the expression matrix contained in the expression set. All these checks can be performed using the *rHVDM* command:

```
> HVDMcheck(fiveGyMAS5)
```

This command returns only warnings and does not modify the object(s) under review. If changes are necessary this is the responsibility of the user.

The pheno data will determine what part of the data set will be used to perform the various HVDM steps. For example, one might want to exclude one of the replicates from the analysis, while keeping intact the pheno data that sits inside the *Biobase* expression set. The relevant *rHVDM* commands will thus accept "external" pheno data. Checking this "external" pheno data is recommended and can also be done using `HVDMcheck`. For example, we can exclude the third replicate from the pheno data

```
> norepl3<-pData(fiveGyMAS5)
> norepl3<-norepl3[norepl3$replicate!=3,]
```

and then check the pheno data is correct:

```
> HVDMcheck(fiveGyMAS5,pdata=norepl3)
```

Note than an `eset` has to be supplied to `HVDMcheck` in any case. As usual, command vignettes are accessible using the question mark:

```
> ?HVDMcheck
```

Before carrying on, let us get rid of what will not be needed:

```
> rm(norepl3)
```

2.2 The training step

In the example data set, we are interested in the transcription factor p53. Before screening genes for p53 dependency, we need to uncover the "hidden" activity profile of p53. This is performed through the *rHVDM* `training` function. A list of known transcription factor targets has to be supplied to the function. In the p53 example, known targets are given in the `p53traingenes` vector, where individual entries are the corresponding gene identifiers.

```
> tHVDMp53<-training(eset=fiveGyMAS5,genes=p53traingenes,
+                                    degrate=0.8,actname="p53")
```

It is very strongly recommended to supply, via the `degrate` argument, an independently measured degradation rate (the importance of this will be illustrated below). Note that *this degradation rate must apply to the first gene in the training set.* So if the degradation rate applies to another gene in the training set, the vector supplied to the `genes` argument should be re-ordered accordingly. The `actname` argument is not compulsory but will help make the reports clearer to read. It is also possible to fit the model using only a subset of the replicates and/or time points. This should be done by supplying an appropriately formatted data frame to the `training` command via the `pdata` argument.

To examine the `tHVDMp53` object that has been generated by the `training` function use:

```
> HVDMreport(tHVDMp53)
```

This command creates an HTML-formatted report and places it *in the working directory* so if one wants the report to be placed in a specific directory, this needs to be changed in the R environment before running the `HVDMreport` command. By default, `HVDMreport` generates a default name for the report, a user-specified name can be assigned using the `name` argument, for example

```
> HVDMreport(tHVDMp53, name="myreport")
```

will generate a document named `myreport.HTML` in the current working directory. Documents generated by `HVDMreport` can be opened and read using standard web browsers. The report for the training step comprises four sections

1. **Training parameters.**

   This part of the report lists all the parameters that were supplied to the `training` function (list of training genes, expression set used, etc.).

2. **Model score.**

   The first graph in the panel compares the model score to the *chi-squared* distribution with an appropriate number of degrees of freedom (*degrees of freedom = data count - parameter count*). The model score is signalled with red crosses. If they are too much to the right of the distribution, the model score is too high which means that the data is not well fitted: one or more of the genes is probably not a target, or is coregulated by another transcription factor; it is also possible that the standard deviation for the measurement errors have been underestimated. Alternatively, the crosses can be too much to the left which signify overfitting, in which case, the standard deviations supplied are probably too high. The case under review shows a desirable outcome. Below, we will give an example where the outcome is not so good.

   The right panel shows a quantile-quantile plot of the standardised deviations between the model and the data with respect to a standard Gaussian distribution. Points should be aligned with the diagonal (as they are in the demonstrating analysis).

   The series of three plots below show the contributions of individual genes and individual time course (normalised by the number of time points in the third plot). This is to help identify where problems may have occurred, for example if a gene contributes disproportionately to the total model score the corresponding vertical bar will be higher than the others. The last series of plots shows the contribution of individual time points within time courses.

3. **Parameter fit.**

The first three figures in this section show a graph of the fitted kinetic parameters, along with a 95% confidence interval. Note that if the hessian matrix is singular, then these confidence intervals cannot be determined, and it is better to assume the worst case scenario, *ie* that they are very large. The second series of graphs shows the transcription factor activity. Note that the first time point (at zero hours) is not fitted and set by default to zero. This means that we are in fact measuring the *deviation from equilibrium* of the transcription factor activity.

The Hessian matrix is both instrumental in the optimising procedure and useful in computing confidence intervals. The next figure in the fit shows the distribution of the eigenvalues of the Hessian matrix. This is useful in evaluating the quality of the fit. If there is a dominant eigenvalue (which is not the case here), then this tells us something about the quality of the fit. Below we will show an example of such an instance.

Finally, The inverse of the Hessian matrix is an approximation of the covariance matrix of the parameters (which is why one can estimate individual confidence intervals) and from the covariance matrix, the correlation matrix can be computed. The correlation matrix is in the last part of the section. In some situations it might be useful to know whether the parameters that are being fitted are dependent upon one another. For example a high basal rate of one particular gene might be compensated by a higher sensitivity rate without affecting the quality of the fit (ie the model score). Such an instance can be seen in the last section of the table, for the gene `209295_at`.

4. **Comparison of model and data.**

   The last part of the report provides a visual comparison between the model (in red) and the data (black). This further allows assessment of the quality of the fit and single out rogue genes and/or replicates. The errors bars indicate the 95% confidence interval for the measurement error, as provided by the user.

The report document contains hyperlinks to facilitate navigation. Note that it is possible to extract more information from the object that has been generated by the `training` function, this is explained in the third section of the present document. We will now present two different training steps where things go slightly wrong to learn how to read these reports. First, we will try to run the training step without anchoring the model. This is done simply by leaving the optional `degrate` argument empty:

```
> tHVDMp53na<-training(eset=fiveGyMAS5,genes=p53traingenes,actname="p53")
```

```
> HVDMreport(tHVDMp53na)
```

Contrasting this report with the previously generated one shows that anchoring the fitting can be quite beneficial to the quality of the fit. There is not much difference in terms of model score (second section of the report), the non-anchored version is even

slightly better, the model score is lower. Equally, the fourth section shows that the data is well fitted by the model. The real difference can be seen in the third section (parameter fit). One can readily see that the confidence intervals are really large both for the kinetic parameters and, especially, the activity profiles. One can also notice that there is a single dominant eigenvalue which indicates that the Hessian is badly conditioned and hence, close to singularity. In this particular case the non-anchored solution is not too far off the anchored one but this is purely coincidental. In any case, confidence intervals that are too large should be avoided.

We now try a final training step, where things go decidedly wrong. Here, we introduce a rogue gene in the training set that is not a p53 target. The affy identifier `202688_at` corresponds to the tumor necrosis factor member 10 (TNF10), but it is a ligand. The real p53 target is the TNF10 *receptor* (and is already part of the training set)[7].

```
> tHVDMp53b<-training(eset=fiveGyMAS5,genes=c(p53traingenes,"202688_at"),
+                                    degrate=0.8,actname="p53")
```

```
> HVDMreport(tHVDMp53b)
```

Problems can can be spotted in many places in this instance. Firstly, the model score is way too high (section 2). Secondly, the parameter values of the newly introduced gene are absurd (section 3). The hessian is singular (section 3). The model visibly struggles to fit the data for this particular gene (last panel of section 4). The fact that it is this particular gene that causes problems can also be seen in the second panel of section 2 of the report: this gene contributes to the total model score in a disproportionate manner.

Before moving on the the next section we can delete unwanted results of some training steps we performed above (*ie* we keep only the `tHVDMp53` object).

```
> rm(tHVDMp53na,tHVDMp53b)
```

2.3 Fitting individual genes and screening in batches

Now that the training step has been performed, we have all the ingredients to start screening the rest of the transcripts. The activity profiles we have computed in the previous section are contained in the `tHVDMp53`. To try individual genes, the `fitgene` function can be used.

```
> gHVDMCD38<-fitgene(eset=fiveGyMAS5,gene="205692_s_at",tHVDM=tHVDMp53)
```

The minimal input to the `fitgene` function are, as above, the expression set(`eset`), the gene name (`gene`) and the output of the `training` function (`tHVDM`). Other possible inputs to the function are a first guess for the parameters. This `firstguess` argument

[7]Note that because in this case things are problematic, the algorithm takes a long time to find an optimum.

should be either a vector with values for the basal rate, the sensitivity and the degradation rate (in that order), alternatively one can give as input to that argument the output to a previous `fitgene` step (it does not even to be the same gene):

```
> gHVDMCD38<-fitgene(eset=fiveGyMAS5,gene="205692_s_at",tHVDM=tHVDMp53,
+                                    firstguess=gHVDMCD38)
```

However, the default setting consists of leaving the `firstguess` argument empty. In that case a first guess is generated internally and, in our experience, this seems to work fine. For example, the second `fitgene` step above is superfluous. To examine the outcome of this individual gene fitting step, the `HVDMreport` can again be used.

```
> HVDMreport(gHVDMCD38)
```

The sections of the HTML report are essentially the same as for the training step. Where possible, comparisons with genes part of the training step are included. The gene under review (CD38/205692_s_at) is very likely to be a p53 target because the model score is fairly low and the sensitivity Z-score is high. It is worth noting that this is a previously unknown p53 target and its dependency status was confirmed using an independent experiment [1]). We can try to fit another (non-p53 target) gene, we pick TNF10l, which we encountered previously:

```
> gHVDMtnf10l<-fitgene(eset=fiveGyMAS5,gene="202688_at",tHVDM=tHVDMp53)
```

```
> HVDMreport(gHVDMtnf10l)
```

We already know that this particular gene is not a p53 target and this shows in the report: the model score is very high, the sensitivity Z-score cannot even be computed because the Hessian is singular (in this case, a default value of -1 is given) and finally, absurd values for the kinetic parameters are returned.

It is also possible to screen in the same fashion a batch of genes using the `screening` function (to screen all the genes in the list remove the square brackets).

```
> sHVDMp53<-screening(eset=fiveGyMAS5,genes=genestoscreen[1:5],HVDM=tHVDMp53)
```

The inputs for this function are more or less the same as for the individual gene fitting command, except that a first guess option is not available. The `genes` argument accepts vectors of gene identifiers. The `genestoscreen` instance is a list of $\approx$ 750 genes we pre-selected that are significantly upregulated and/or present at at least one point in time in the context of this experiment. It is worth to avoid scanning the whole genome. Although fitting an individual gene can be done in a few seconds, those seconds add up if 20k genes are screened! The above example should take approximately five minutes on a standard personal computer (as of December, 2006).

Other arguments to the `screening` function are the various thresholds beyond which individual genes will be considered to be targets of the transcription factor under review.

The `cl1zscorelow` argument is the bound above which a given sensitivity Z-score will classify a gene as a putative transcription factor target. Likewise, the `cl1modelscorehigh` is the bound for the model score. To be considered a target, the transcript must have a model score *below* that bound. Finally, the `cl1degraterange` is the range (given as two entries vector) within which the fitted degradation rate must find itself. To be considered a target, a given gene must satisfy all three conditions. Default values are respectively (2.5,100.0,c(0.05,5.0)). A list of putative p53 targets (collated in a data frame) can, in our example, be obtained via the command

```
> p53targets<-sHVDMp53$results[sHVDMp53$results$class1,]
```

The field `class1` in the data frame flags (TRUE/FALSE) the dependency status of each individual gene. The thresholds values can be changed by running the screening command again with revised bounds. Imagine for example that we want to be more restrictive both on the model score and sensitivity Z-score criteria. This can be done by plugging in the new bounds and in place of the training object, supplying the previously obtained screening object:

```
> sHVDMp53<-screening(HVDM=sHVDMp53,cl1modelscorehigh=80.0,
+                              cl1zscorelow=3.5)
```

The new list can be obtained as above. Note that in this new "run" the `eset` and `genes` arguments can be omitted as the original screening object contains all the necessary information. In other words, actual fitting of each gene is not performed once again. The list of genes that are obtained after any screening step are, by default, ordered by descending sensitivity Z-score. We found that, using an independent experiment [1], the sensitivity Z-score is a better predictor of transcription factor dependency, genes at the top the list are the most likely to be actual targets. Finally, the infamous `HVDMreport` function can also be applied to the output of the `screening` function.

```
> HVDMreport(sHVDMp53)
```

The HTML file the function produces provides a wealth of information about the parameters of the original training (section 1) and screening (section 2) steps. In addition, a count of the genes passing the various thresholds is produced (in section 3). A list of putative targets is given in the final, and fourth, section of the HTML document.

3 Accessing information in rHVDM objects

Objects generated by the `training`, `screening` and `fitgene` functions are not objects in the strict R sense but mere `lists` which themselves contain other `lists`, `vectors`, `data.frames` and `character` classes. In this section we describe in some detail the content of these objects in order for the end user to be able to access it and for example to create their own charts etc.. When creating numeric R objects it is always possible

to label individual entries (in the case of a matrix, individual entries can be accessed by their row and column labels). This possibility is fully exploited in *rHVDM* and the labels are designed to be as explicit as possible, which should hopefully make data retrieving relatively transparent.

3.1 List created by the `training()` function.

In R, names of list members can be accessed using the `names` function. Using this function with one of the lists created in the previous section with the `training` function reveal that this lists comprises ten objects: `tc, dm, par, pdata, distribute, results, scores, itgenes, eset, type`. Accessing these members individually can be done as usual using the R operator `$`, *ie* `name_of_list$name_of_member`.

The `dm` member.

This member is itself a list and it contains the data, the model fit for the data and the standard deviation of the measurement error. Each of these three are stored in flat vectors named respectively `signal`, `estimate` and `sdev` and stored as members or `dm`. Each individual entry in these vectors is labelled by a character string comprising the gene name, the name of the experiment, the name of the replicate and the time point, in that order and separated by single dots. Going back to the example of the previous section,

```
> tHVDMp53$dm$signal[paste("202284_s_at","5Gy","2",6,sep=".")]

202284_s_at.5Gy.2.6
          313.5024
```

will return the measured expression at 6 hours for gene `202284_s_at` in replicate `2` of the `5Gy` time course.

The `par` and `distribute` members.

The `par` member is itself a list containing, among others, the values of the parameters of the model. These comprise the three kinetic parameters for each gene and the values of the transcription factor activity for each time point in each time course and replicate. These values are stored in the `parameters` member of `par`. The values are labelled using dot separated labels. For example,

```
> tHVDMp53$par$parameters[paste("p53","5Gy","2",6,sep=".")]

p53.5Gy.2.6
   218.6031
```

accesses the value for the activity of the transcription factor in the 6 hours data point of replicate 2 of the `5Gy` time course. Similarly, individual kinetic parameters can be accessed by specifying the gene identifier (`203409_at`), followed by the kinetic parameter identifier (`Dj`)

```
> tHVDMp53$par$parameters[paste("203409_at","Dj",sep=".")]

203409_at.Dj
   0.2989867
```

Basal, sensitivity and degradation parameters are denoted with `Bj, Sj, Dj`. Some of these parameters are fixed (initial time points of the transcription factor activity profile are set to zero, and one or two parameters for the anchoring gene). A vector containing the values of these fixed parameters can be obtained using

```
> tHVDMp53$distribute$known

202284_s_at.Sj 202284_s_at.Dj    p53.5Gy.1.0    p53.5Gy.2.0
           1.0            0.8            0.0            0.0
   p53.5Gy.3.0
           0.0
```

Conversely, the vector of free parameters *labels* is in the `free` of the `distribute` member, therefore

```
> tHVDMp53$par$parameters[tHVDMp53$distribute$free]

   p53.5Gy.1.2    p53.5Gy.1.4    p53.5Gy.1.6    p53.5Gy.1.8
   202.4416486    365.7310865    210.1534254    141.0537131
  p53.5Gy.1.10   p53.5Gy.1.12    p53.5Gy.2.2    p53.5Gy.2.4
   137.1960408    134.7814629    185.0595864    306.9992291
   p53.5Gy.2.6    p53.5Gy.2.8   p53.5Gy.2.10   p53.5Gy.2.12
   218.6030859    134.4525120    107.2278220    152.1188719
   p53.5Gy.3.2    p53.5Gy.3.4    p53.5Gy.3.6    p53.5Gy.3.8
   231.9349796    344.0454844    271.8306314    153.9791938
  p53.5Gy.3.10   p53.5Gy.3.12 202284_s_at.Bj   203409_at.Bj
   138.9322073    157.5026006      6.6992269     57.9463447
  203409_at.Sj   203409_at.Dj 218346_s_at.Bj 218346_s_at.Sj
     1.6526153      0.2989867     34.1479182      0.4472608
218346_s_at.Dj   205780_at.Bj   205780_at.Sj   205780_at.Dj
     0.4041790     30.3413971      0.6606399      0.5874381
  209295_at.Bj   209295_at.Sj   209295_at.Dj
    12.3201042      0.3892236      0.4345314
```

returns the (labelled) values of these free parameters. Other components of the `par` and `distribute` members are of less direct interest here. These extra components in `distribute` act as an interface between the `training` list and the Levenberg-Marquardt optimiser, while the extra parameters in `par` specify the type of model to be used (we plan to expand *rHVDM* for it to accommodate non-linearities in the response and multiple transcription factor dependencies).

The `scores` member.

The total model score of equation (4) can be broken down by gene, time course etc. These various score ventilations are collated in the `scores` member. The vector `s2` is the atomised version (it contains the breakdown for each possible (gene/ time point/ experiment/ replicate). The labelling of this vector is exactly the same as the one used in the `dm` member. The scalar `total` contains the total model score. Between these two extremes, `bygenes` is the distribution by gene, `bytc` by time course (a time course is an experiment/replicate pair). `bytcpertp` is also the distribution by time course, but divided by the number of points in the time course. Within time course score distributions per time point are also provided. Since the number of time courses is not predictable, the space to store these results is allocated dynamically in the `withintc` list. Each (numbered) member of this list is itself a list that contains the time course label (in the `name` slot) and the the score breakdown per time point (in `score` slot). For example,

```
> tHVDMp53$scores$withintc[[1]]$score
```

```
        0         2         4         6         8        10
 1.326517 12.893795  3.258966  3.242109  4.321411  6.388368
       12
 2.471207
```

returns one of these within time course breakdowns.

The `results` member.

This member contains the output of the optimisation step. The `hessian` slot is the hessian at the optimum.

Less interesting member (`tc`).

This list member contains technical information used for solving the model equation, in particular the differential operator A mentioned in (3). This information has to be determined for each time course independently and is dynamically stored in `tc`, which is an indexed R list. Each of these slots is itself a list containing the name of the experiment (`experiment`), replicate (`replicate`) the time values (`time`), the appropriate column

labels to access the data (`explabel`) and the differential operator (`A`). To view the latter for the first time course, type

```
> tHVDMp53$tc[[1]]$A
```

```
           [,1]        [,2]        [,3]        [,4]
[1,]  0.0000000  0.00000000  0.00000000  0.00000000
[2,] -0.4722222  0.25000000  0.25000000 -0.02777778
[3,]  0.2152778 -0.66666667  0.25000000  0.22222222
[4,]  0.0000000  0.04166667 -0.33333333  0.00000000
[5,]  0.0000000  0.00000000  0.04166667 -0.33333333
[6,]  0.0000000  0.00000000  0.00000000  0.08333333
[7,]  0.0000000  0.00000000  0.00000000  0.00000000
            [,5]        [,6]        [,7]
[1,]  0.00000000  0.00000000  0.00000000
[2,]  0.00000000  0.00000000  0.00000000
[3,] -0.02083333  0.00000000  0.00000000
[4,]  0.33333333 -0.04166667  0.00000000
[5,]  0.00000000  0.33333333 -0.04166667
[6,] -0.50000000  0.25000000  0.16666667
[7,]  0.25000000 -1.00000000  0.75000000
```

Least interesting members (`pdata, eset, type, itgenes`).

The remaining list members contain bookkeeping information. The `pdata` member is simply a `data.frame` objet that is a copy of the pheno data that being used by the `training` function and is either the pheno data of the expression set or, the alternative pheno data fed to that function by the user. The `eset` member is the name of the Biobase expression set and `type` indicates what kind of object we are dealing with. At the end of each training set, a "mini-screening" is performed with each member of the training set for subsequent comparison purposes. The results of this screening are contained in the `itgenes` member (a `data.frame`), whose formatting is similar to the one produced by the `screening` command (see subsection 3.3).

3.2 List created by the `fitgene()` function.

The object generated by the `fitgene` function are structured exactly in the same manner as the ones generated by the `training` and therefore contains the same members (refer to the previous section, the member `tset` is a copy of the `tset` member of the training object) plus a few extra ones. These are:

- `fguess`: the first guess generated inside the function or, alternatively, the one supplied by the user.

- `tHVDMname`: the name of the object generated by the `training` used in the `fitgene` function.

3.3 List created by the `screening()` function.

The list created by the `screening` function comprises five members: `results, tHVDM, transforms, class1bounds, type`. The last four contain essentially parameters that were input to the function:

- `tHVDM`: is a straight copy of the `training` object (see subsection 3.1).
- `transforms`: indicates what kind of transformation, if any, was applied to the some kinetic parameters, it is a vector whose character entries indicate the transformed kinetic parameter. The function used for the transform is indicated in the labels.
- `class1bounds` is a list collating the bounds governing whether individual genes are classified as putative targets of the transcription factor of interest or not. The member `zscore` is a lower bound for the sensitivity Z-score. If a gene's sensitivity Z-score is below this value, it will not be considered a target. The `modelscore` is also a scalar, but it is an *upper* bound. Genes with a model score *above* this value will not be considered targets. Finally, the `degrate` member of `class1bounds` is vector containing the lower and upper bounds for the degradation rate.

The most important member of this list, however is the `results` member. It is a `data.frame` object in which each record collates the essential results of the screening for each individual gene. This data frame comprises 14 columns:

- `model_score`: The model score for the gene.
- `sens_z_score`: The Z-score for the sensitivity.
- `basal`: The basal transcription rate for the gene
- `sensitivity`: The sensitivity to the transcription factor activity.
- `degradation`: The degradation rate for the gene.
- `basal_d, basal_u`: Respectively the lower and upper value for the 9% confidence interval of the basal rate.
- `sensitivity_d, sensitivity_u`: Same as above, for the sensitivity.
- `degradation_d, degradation_u`: Same as above, for the degradation.
- `class1`: A boolean (True/False) specifying wether the gene belongs to the list of putative targets.

- `hess_rank`: The rank of the hessian matrix. If this is lower than 3, then the system is singular and the corresponding gene is not a putative target (the confidence intervals and Z-scores cannot be computed in this case).
- `lm_message`: Message returned by the optimising procedure.

4 Technical issues

4.1 How confidence intervals are evaluated

As has been mentioned previously in this document, the Hessian at the optimal point, as found by the Levenberg-Marquardt procedure(LM), is used to compute confidence intervals. This sounds straightforward but some peculiarities in our fitting algorithm make matters slightly more complicated. When fitting the model, we advocate[8] forcing some parameters in the positive domain. This is done by feeding the LM procedure with parameters that are "free to roam" everywhere in the parameter space (including negative portions of it). Before evaluating the model, however the exponential of these "free" values is taken, forcing them to be positive. Another way of putting this is that the values used in LM are log-transforms of the actual parameters values. Before looking at this case, we examine the more straightforward situation where there is no forcing. We assume that the hessian (`H`) has full rank and the vector of free parameters is denoted by `v` (the way to extract these two objects is explained in section 3.1). The confidence interval bounds could then be obtained using

```
> Vcov<-solve(H)          #the variance-covariance matrix is obtained
                          #by inverting the hessian
> halfint<-1.96*diag(Vcov)^0.5 #Taking the square root of the diagonal
                               #of that matrix gives the standard
                               #deviation, multiplying by 1.96 gives
                               #the 95% half confidence interval.
> upperbound<-v+halfint #upper bound for c.i.
> lowerbound<-v-halfint #lower bound for c.i.
```

In the case mentioned above, the hessian is valid for (log-)transformed values (for some parameters) and, consequently the confidence intervals are first computed in this domain and then lifted in the non-transformed domain we are interested in. This is achieved by running the following set of commands (we take as an example the training outcome shown in the sample session in section 2):

```
> vcov<-solve(tHVDMp53$results$hessian)
> sdev<-1.96*diag(vcov)^0.5 #compute half confidence interval in
```

[8] It is, however, possible to relax this, see vignettes of the `training`, `screening` and `fitgene` functions.

```
>                          #transformed domain
> work<-tHVDMp53 #create a copy of the example to work with
> central<-.exportfree(work) #extract transformed, fitted values
> nams<-names(central) #extract vector of labels
> upperbounds<-
+   (.importfree(HVDM=work,x=central[nams]+sdev[nams]))$par$parameters
>   #.importfree() imports the transformed values for the upper bound
>   #into the "work" object and performs the inverse transform (here, an
>   #exponential) upon import, a new HVDM object is returned by this
>   #command. The $par$parameters suffix extracts the parameter vector.
>   #Note that the upperbounds vector will contain all parameter values,
>   #ie not only those fitted in LM.
>
> lowerbounds<-
+   (.importfree(HVDM=work,x=central[nams]-sdev[nams]))$par$parameters
>   #a similar command is used to extract the lower bounds.
>
> rm(work) #get rid of this no longer needed object
```

Note that this set of command will also work if no transform was used during the fit.

5 Implementation notes and other practicalities

5.1 MacOSX 10.4

- To run properly, the `HVDMreport` should have an X11 device. To do that in the R session, select *run X11 server* from menu *Misc*.

- *rHVDM* requires *minpack.lm* which in turn requires a FORTRAN compiler. Instructions can be found here: `http://cran.r-project.org/bin/macosx/RMacOSX-FAQ.html#Fortran-compiler-_0028g77-3_002e3-or-later_0029`. For this particular context, there does not need to be a match between the C and FORTRAN compiler, as only the latter is used. Alternatively, binaries for the *minpack.lm* and *R2HTML* packages can be found on the CRAN website `http://cran.r-project.org/`. The Bioconductor packages and clear instructions on how to install them can be found on the Bioconductor website `http://www.bioconductor.org/docs/install-howto.html`.

5.2 Windows

- To install *rHVDM*, we recommend using the binaries available on the bioconductor website.

- Windows binaries for two of the required packages (*minpack.lm* and *R2HTML*) can be found on the CRAN website `http://cran.r-project.org/`. The Bioconductor packages and clear instructions on how to install them can be found on the Bioconductor website `http://www.bioconductor.org/docs/install-howto.html`.

6 What's new in version...

version 1.1

- Updated the package so that it only accepts the new `ExpressionSet`. The old `exprsSet` is deprecated (as in the rest of the Bioconductor project).
- Vignettes and the present textual description are modified accordingly, with some indications on how to transform the old input objects into new ones.
- Corrected some typos in the present document.

version 1.2-1.4

- version 1.2 was the release version of 1.1.
- vesion 1.3 and 1.4 are identical to 1.2 (the numbers were bumped automatically because of a Bioconductor update.

version 1.5

- Computation of the measurement error for selected plattforms.
- Update of the present document and vignettes.

version 1.6

- Revert to older version (too many bugs in the error computation feature).

version 1.7.5 (devel) and 1.8.0 (release)

- Re-introduction of computation of the measurement error.

version 1.8.1 and 1.9.1

- Introduction of more axes labelling in the reports.

versions 1.9.2

- Introduction of NAMESPACE feature

versions 1.9.3

- Introduction of a new plattform for the measurement error estimation (affy HG ST1.0 array.)

versions 1.9.4

- Introduction of new commands and data to accomodate a nonlinear formulation. Individual vignettes are written. A full documentation will be done in susequent version.)

versions 1.9.5

- Changed the way the plattform-specific parameters are stored within the package.

Future updates

- On-line generation of diagnostic graphs (to complement off-line HTML report generation).
- Nonlinearities in the production function (mainly saturation but also threshold effects).
- Repression
- Multiple transcription factors dependencies.
- Extension of the measurement error computation to more plattforms.

References

[1] M. Barenco, D. Tomescu, D. Brewer, R. Callard, J. Stark, and M. Hubank. Ranked predictions of p53 targets using hidden variable dynamic modelling (hvdm). *Genome Biology*, 7(3):R25, 2006.

[2] R. C. Gentleman and et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biol*, 5(10):R80, 2004.

[3] Eric Lecoutre. The R2HTML package. *R News*, 3(3):33–36, December 2003.

[4] M. Barenco, J. Stark, D. Brewer, D. Tomescu, R. Callard, and M. Hubank. Correction of scaling mismatches in oligonucleotide microarray data. *BMC Bioinformatics*, 7:251, 2006. 1471-2105 (Electronic) Journal Article.

[5] J. Comander, S. Natarajan, Jr. Gimbrone, M. A., and G. Garcia-Cardena. Improving the statistical detection of regulated genes from microarray data using intensity-based variance estimation. *BMC Genomics*, 5(1):17, 2004. 1471-2164 Journal Article.

[6] A. Kamb and M. Ramaswami. A simple method for statistical analysis of intensity differences in microarray-derived gene expression data. *BMC Biotechnol*, 1(1):8, 2001. 1472-6750 Journal Article.

[7] D. M. Rocke and B. Durbin. Approximate variance-stabilizing transformations for gene-expression microarray data. *Bioinformatics*, 19(8):966–72, 2003. 22645880 1367-4803 Journal Article.