



# Using RChip

Aloke Phatak  
5 June 2007

Enquiries should be addressed to:

Aloke Phatak ([Aloke.Phatak@csiro.au](mailto:Aloke.Phatak@csiro.au))

Harri Kiiveri ([Harri.Kiiveri@csiro.au](mailto:Harri.Kiiveri@csiro.au))

## **Important Notice**

© Copyright Commonwealth Scientific and Industrial Research Organisation

('CSIRO') Australia 2007

All rights are reserved and no part of this publication covered by copyright may be reproduced or copied in any form or by any means except with the written permission of CSIRO.

The results and analyses contained in this Document are based on a number of technical, circumstantial or otherwise specified assumptions and parameters. Users must make their own assessment of the suitability of the information or material contained in or generated from the Document. To the extent permitted by law, CSIRO excludes all liability to any party for expenses, losses, damages and costs arising directly or indirectly from using this Document.

# Using RChip

Aloke Phatak

June 5, 2007

## 1 Introduction

RChip is a suite of functions for analyzing the relationship between response variables and a set of predictors when the number of predictors far exceeds the number of observations. Such data are commonly observed in gene expression studies, where the expression levels of tens of thousands of genes are measured from samples taken from tens or a few hundred individuals. Other examples include data from SNP (single nucleotide polymorphism) arrays and from mass spectrometers. Although the functions in RChip were originally conceived for analyzing data from bioinformatics studies, they are not of course limited to such data and can be used in many other contexts where the ' $N < p$ ' problem restricts the scope of conventional model-building and parameter estimation procedures.

Underlying many of the functions in RChip is an engine that provides the mechanism for eliminating redundant variables in a wide variety of existing statistical models, such as generalized linear models, multiclass logistic regression, and proportional hazards survival models. This mechanism is based on the notion of model *sparsity*, and hence allows for sensible estimation of parameters when  $N$ , the number of observations, is much less than  $p$  the number of potential explanatory variables. Details of the basic methodology may be found in Kiiveri (2003)<sup>1</sup>

The library contains functions for simultaneous variable selection and parameter estimation in several commonly used procedures, but in this vignette, we focus on one of the most common applications: the analysis of gene expression data where the objective is to identify genes that discriminate between different phenotypes. We describe the use of the function `HGmultc`, which fits a multiclass logistic regression model and which, at the same time, eliminates redundant explanatory variables.

If, for example, we use the symmetric multiple logistic transformation defined by Friedman *et al.*, (2000)<sup>2</sup>, then the probability that a sample will be in class  $g$  is given by

$$p_g = \frac{\exp(\mathbf{x}'\boldsymbol{\beta}_g)}{\sum_{h=1}^G \exp(\mathbf{x}'\boldsymbol{\beta}_h)} \quad g = 1, 2, \dots, G \quad (1)$$

Here,  $\mathbf{x}$  is a  $p$ -vector containing the measurements of expression levels for that sample, and  $\boldsymbol{\beta}_g$  is a set of  $p$  coefficients, one for each probeset on the array, for class  $g$ . A training data set is used to estimate the coefficients  $\boldsymbol{\beta}_g$  in a Bayesian framework, and they are obtained by maximizing the posterior distribution of the weights given the training data. One of the key

---

<sup>1</sup>Kiiveri, H. (2003) A Bayesian approach to variable selection when the number of variables is very large. In *Science and Statistics: A Festschrift for Terry Speed*, D. R. Goldstein (ed.), IMS Lecture Notes - Monograph Series, Volume 40, Institute of Mathematical Statistics, Beachwood, Ohio.

<sup>2</sup>Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. *Ann. Statist.*, **28** (2), 337–407.

assumptions of the engine underlying RChip, which is embodied in the prior distributions assigned to the vectors of coefficients  $\beta$ , is that most of the coefficients are zero. Hence, we obtain very parsimonious models that include only a few genes and that are easily interpretable.

## 2 Using HGmultc

### 2.1 St. Jude's Leukemia Data

We illustrate the use of HGmultc using the leukemia data published by Yeoh *et al.* (2002)<sup>3</sup> and available as a data package (stjudem<sup>4</sup>) from the Bioconductor repository. The data package is also associated with macat<sup>5</sup>, a Bioconductor package containing methods for identifying differentially expressed chromosome regions.

If you have root privileges on a Unix machine or administrator privileges on a Windows PC, the data package can be installed by issuing the command

```
> install.packages(pkgs = "stjudem",
+   repos = "http://bioconductor.org/packages/1.9/data/experiment")
```

Alternatively, download the compressed file directly from Bioconductor, and then install it as follows:

```
> install.packages(pkgs = "/PATH/TO/FILE/stjudem_1.2.0.tar.gz",
+   repos = NULL)
```

To load the data, issue the command

```
> library(stjudem)
```

The data package contains a single data frame called stjude, and it contains the following elements:

```
> names(stjude)

[1] "geneName"      "geneLocation"
[3] "chromosome"   "expr"
[5] "labels"       "chip"
```

The matrix of gene expression values is contained in the object stjude\$expr. Its columns are samples and its rows probesets, but because HGmultc requires the transpose, we construct X, the matrix of explanatory variables, as follows:

```
> X <- t(stjude$expr)
> dim(X)

[1] 327 12625
```

Hence, the data consists of 327 arrays, each containing 12 625 probesets. For details on the pre-processing of raw expression levels, see the original article. The arrays have been classified into 10 disease categories, whose labels are given by

---

<sup>3</sup>Yeoh, E.-J., Ross, M.E., and 19 others. (2002). Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling. *Cancer Cell*, 1, 133–143.

<sup>4</sup><http://bioconductor.org/packages/1.9/data/experiment/html/stjudem.html>

<sup>5</sup><http://bioconductor.org/packages/1.9/bioc/html/macat.html>

```
> table(stjude$labels)
```

BCR	E2A	Hyperdip	Hyperdip47
15	27	64	23
Hypodip	MLL	Normal	Pseudodip
9	20	18	29
T	TEL		
43	79		

We can see that the dataset is somewhat unbalanced, that some classes have many more samples than others. In demonstrating the use of `HGmultc` below, we will exclude the classes `Hyperdip47`, `Hypodip`, and `Pseudodip`.

## 2.2 Running `HGmultc`

The function `HGmultc` has only two required arguments: a *matrix* of expression values  $X$ , where the  $N$  rows correspond to samples or arrays, and the  $p$  columns to probesets, and an  $N$ -element *numeric vector* of class labels  $y$ :

```
> args(HGmultc)
```

```
function (X, y, weights = rep(1, nrow(X)), sparsity.prior = "NG",
  lambda = 1, bbess = 1e+07, kbess = 0, adj.prior = TRUE, control = HGcontrol())
NULL
```

If there are  $G$  classes, then the vector of class labels must contain the integers  $1, 2, \dots, G$  corresponding to the classes of the rows of  $X$ . If  $G = 2$ , for example, the class labels would have to be specified as  $(1, 1, 1, 2, 2, 2, 2)$ , but *not* as  $(2, 2, 2, 5, 5, 5, 5)$ . For the leukemia data, the vector of labels can be constructed as

```
> y <- rep(1:7, times = table(stjude$labels)[-c(4,
+ 5, 8)])
```

where the classes identified above have been removed. Consequently, we have to redefine the matrix  $X$  to remove observations from these classes:

```
> Index <- rep(1:10, times = table(stjude$labels)) %in%
+ c(1:3, 6:7, 9:10)
> X <- X[Index, ]
```

and  $X$  now has 266 rows.

The default values of the parameters `kbess` and `bbess` work well in most circumstances, so let's use them here to run `HGmultc`:

```
> res.HGmultc <- HGmultc(X, y)
```

The result is an object of class '`HGmultc`', and it contains the following elements:

```
> names(res.HGmultc)
```

```
[1] "beta" "S" "P" "class"
```

where

beta is a  $(p + 1) \times G$  matrix of parameter estimates, *most of which are zero*. There is one column per class, and its elements are the estimates of the elements of  $\beta_h$  in eq. (1).

S is a  $(p + 1) \times G$  logical matrix. Its elements are TRUE if the variable is selected, i.e., if the corresponding estimate of  $\beta_{ij}$ ,  $i = 1, 2, \dots, p$ ,  $j = 1, 2, \dots, G$ , is non-zero.

P is an  $N \times G$  matrix of fitted probabilities,  $\hat{p}_{hj}$ ,  $h = 1, 2, \dots, N$ ,  $j = 1, 2, \dots, G$ . The rows of P sum to one, i.e.,  $\sum_{j=1}^G \hat{p}_{hj} = 1$ .

class is an  $N \times 1$  vector of fitted class labels obtained by identifying the class in each row of the matrix P that yields the largest fitted probability.

To see the which probesets have been chosen for each class, and their corresponding parameter estimates,

```
> VarEst <- apply(res.HGmultc$beta *
+   res.HGmultc$S, 2, function(x) {
+   x[x != 0]
+ })
> names(VarEst) <- unique(stjude$labels)[-c(4,
+   5, 8)]
> lapply(VarEst, round, 2)
```

\$BCR

	1635_at	37600_at	39318_at
-61.32	36.29	6.07	-1.29
41872_at			
11.27			

\$E2A

	430_at
-20.27	10.76

\$Hyperdip

	34593_g_at	37014_at	38968_at
-1.97	-1.78	3.99	4.58

\$MLL

	33412_at	36638_at	38291_at
-9.34	3.68	2.28	2.54

\$Normal

	35939_s_at	37645_at	39878_at
6.75	-4.23	3.21	4.10
40113_at			
-5.85			

\$T

	32794_g_at	39318_at
-16.01	10.48	-2.21

```
$TEL
      1488_at 39389_at 41097_at
-22.15    10.76    -3.54     6.60
```

So, we can see that from a dataset consisting of 266 observations and 12 625 variables, `HGmultc` has selected only 20 probesets that are able to distinguish between these 7 classes. The ‘plug-in’ misclassification error is given by

```
> Y <- as.factor(rep(names(VarEst),
+   times = table(y)))
> table(Y, res.HGmultc$class)
```

Y	1	2	3	4	5	6	7
BCR	15	0	0	0	0	0	0
E2A	0	27	0	0	0	0	0
Hyperdip	0	0	62	0	2	0	0
MLL	0	0	0	20	0	0	0
Normal	0	0	0	0	18	0	0
T	0	0	0	0	0	43	0
TEL	0	0	0	0	0	0	79

and though the model is very parsimonious, classification is perfect! Of course, plug-in estimates tend to be optimistic, and we can obtain the cross-validated misclassification table by using the function `xvalidate`, with  $q$ -fold crossvalidation. A word of caution here: depending on the speed of the CPU and the amount of RAM, and the value of  $q$ , cross-validation can take considerable time! To carry out ten-fold cross-validation, we issue the command

```
> res.xval <- xvalidate(X, y, method = HGmultc,
+   fold = 10, trace = FALSE)
```

and the result, `res.xval`, is a vector of cross-validated fitted values. The resulting misclassification table is given by

```
> table(Y, res.xval)
```

Y	res.xval						
	1	2	3	4	5	6	7
BCR	9	0	3	1	2	0	0
E2A	0	27	0	0	0	0	0
Hyperdip	3	0	56	1	2	1	1
MLL	0	1	1	15	3	0	0
Normal	0	0	8	0	9	0	1
T	0	0	0	1	0	42	0
TEL	0	0	0	0	1	0	78

The cross-validated misclassification error is larger than the corresponding plug-in quantity, but, with the exception of the classes `BCR` and `Normal`, the error is still relatively small. Note that in the original paper, the `Normal` group was shown to be quite heterogeneous, so the relatively large misclassification error is not surprising. The classification of individuals in the `BCR` group might be improved by passing the argument `control = HGcontrol(tolc = 1e-04)` to the function `xvalidate`.

### 3 Visualizing the Results of `HGmultc`

If only relatively few variables chosen by `HGmultc` – unlikely when there are several classes – then it is possible to visualize the separation of the classes by constructing pairwise plots of the probesets that have been selected. When there are more than say 4–6 variables selected, we can still view the separation of the classes in few dimensions by plotting the observations in the space of the first few linear discriminant functions<sup>6</sup>. Linear discriminant functions are linear combinations – in this case, of the variables (probesets) selected by `HGmultc` – that best separate the classes.

We first extract the probesets selected by `HGmultc`,

```
> x <- X[, unlist(apply(res.HGmultc$S,  
+ 2, function(x) {  
+   which(x)[-1] - 1  
+ }))] ]
```

and then, after running the function `lda` in the library `MASS`, save the discriminant functions into a data frame in preparation for plotting them using `lattice` graphics.

```
> require(MASS)
```

```
[1] TRUE
```

```
> res.lda <- lda(x, y)  
> z <- scale(x, scale = FALSE) %*% res.lda$scaling  
> z <- as.data.frame(z)
```

Figure 1 below shows the results of plotting observations in the space of the first three discriminant functions, and we can clearly see the separation of the classes.

### 4 A General Strategy for Running `HGmultc`

As we pointed out earlier the default values of the hyperparameters `kbess` and `bbess` work well in most situations. Nevertheless, the user may wish to find ‘optimal’ values by calculating cross-validated error rates over a grid indexed by a range of values of the hyperparameters. Such a procedure is, not surprisingly, computationally intensive. For `HGmultc`, it can be carried out as follows:

```
> res.array <- optim.hyperpars(X, y,  
+   method = HGmultc, xvalid = TRUE,  
+   fold = 10, trace = TRUE, kbess = seq(0,  
+     1, by = 0.1), bbess = 10^seq(0,  
+     7))
```

Here, cross-validated error rates are carried out using ten-fold cross-validation over a grid indexed by values of `kbess` and `bbess` in the ranges  $(0, 0.1, \dots, 1)$  and  $(10^0, 10^1, \dots, 10^7)$ , respectively. The result is a list with a single component `xtable`, which in this instance is an  $11 \times 8$  array of cross-validated error rates. In most circumstances, testing a range of values of `kbess` is sufficient because it usually has the biggest effect on cross-validation error.

---

<sup>6</sup>Venables, W.N and Ripley, B.D. (2002). *Modern Applied Statistics with S*, 4th edition. Springer-Verlag: New York. See Chapter 12.1



```
> require(lattice)
```

```
[1] TRUE
```

```
> print(cloud(LD3 ~ LD2 * LD1, data = z,  
+ groups = Y, key = list(text = list(levels(Y)),  
+ points = Rows(trellis.par.get("superpose.symbol"),  
+ 1:7), columns = 3)))
```

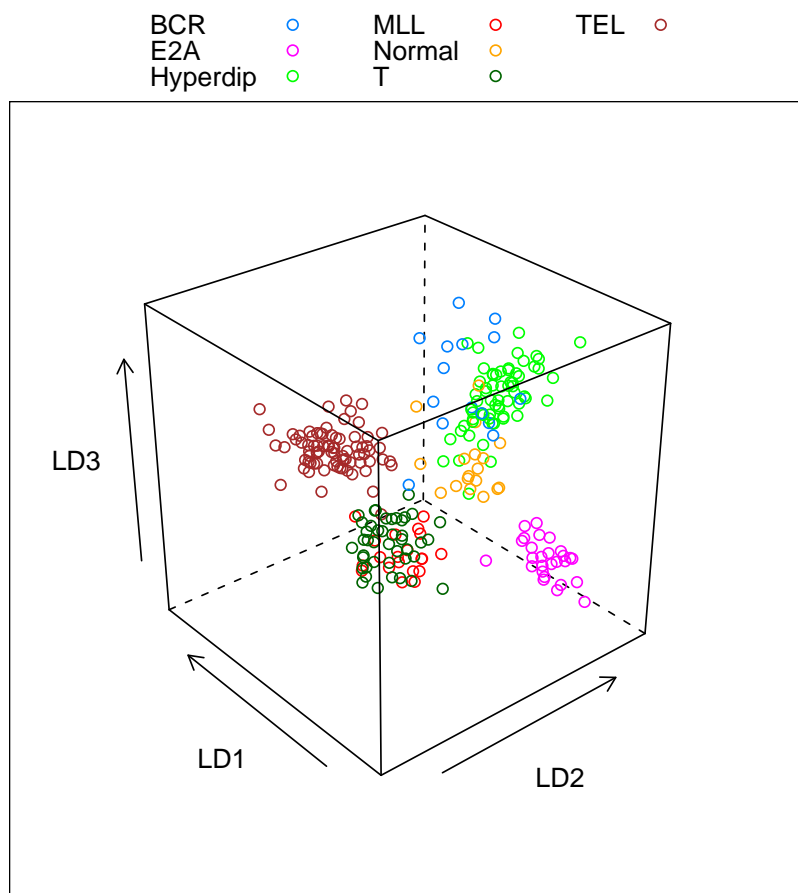


Figure 1: Plot of the leukemia data in the space of the first three linear discriminant functions derived from the probesets selected by HGmu1tc.

## 5 Sparse Generalized Linear Models

The parameter elimination and fitting routine at the heart of `HGmultc` has been used to construct an engine that provides the mechanism for eliminating redundant variables in a much broader class of statistical models than multiclass logistic regression. We briefly discuss one such function – `HGglm` – that makes use of the general fitting routine.

The function `HGglm` fits generalized linear models (GLM) with sparsity priors, and it carries out simultaneous variable selection and parameter estimation within a wide class of models that includes logistic regression, Poisson regression, GLMs with gamma-distributed observations, and others. More importantly, however, `HGglm` allows users to specify their own models. As we shall illustrate below, the additional arguments to `HGglm` are derived from the specifications required to define any GLM, namely the log-likelihood and link function.

The syntax of the call to `HGglm` is as follows:

```
> res.HGglm <- HGglm(x, f, event = NULL,  
+ weights = rep(1, nrow(x)), sparsity.prior = "NG",  
+ bbess = 1e+07, kbess = 0, b0sc = 15,  
+ scale = -1, initb = "FALSE", model = "N",  
+ fvalfn = NULL, ifvalfn = NULL,  
+ varfn = NULL, drfn = NULL, devfn = NULL,  
+ scale.updatefn = NULL, no.prior = 1)
```

For a detailed explanation of all the arguments, see the help file for `HGglm`. In the current implementation, the possible values of the `model` argument allow for a wide range of common models. Setting `model = "Own"` permits users to specify their own models, and requires further arguments to pass the model specification to the variable selection and fitting engine. To illustrate how to do so, we will implement Poisson regression explicitly and show that it yields similar results to the Poisson regression model that is built-in to `HGglm` and that can be carried out by setting `model = "P"`.

For  $n$  independent Poisson observations  $(y_1, y_2, \dots, y_n)$  with  $E(Y_i) = \mu_i$  and  $\text{Var}(Y_i) = \mu_i$ , the log likelihood can be written as

$$l(\mu_1, \mu_2, \dots, \mu_n | y_1, y_2, \dots, y_n) = \sum_{i=1}^n (y_i \log \mu_i - \mu_i) \quad (2)$$

and we specify the canonical link  $\eta_i = \log \mu_i$  with linear predictor  $\eta_i = \mathbf{x}_i^T \boldsymbol{\beta}$ . Now when `model = "Own"`, the arguments, `fvalfn`, `ifvalfn`, `varfn`, `drfn`, `devfn`, and `scale.updatefn` must be specified, and they are described below:

`fvalfn` a function of the linear predictor  $\eta$  which returns  $\mu$ ;

`ifvalfn` a function of the mean  $\mu$  which returns  $\eta$ ;

`varfn` a function of  $\mu$  which returns the variance function;

`drfn` a function of  $\mu$  which returns the value of  $d\mu/d\eta$ ;

`devfn` a function of  $\mu$  and the data  $y$  (and possibly of the arguments `weight`, `scale`, and `event`) which returns the value of the deviance; and

`scale.updatefn` a function of  $\mu$  and the data  $y$  (and possibly of the arguments `weight`, `scale`, and `event`) which returns the value of the scale parameter.

For the Poisson distribution, there is no scale parameter, but we still need to define the function `scale.updatefn`, albeit as a function that returns only a constant value. Hence, we have

```
> fvalfn.p <- function(eta, y, event,
+   weights, scale) {
+   exp(eta)
+ }

> ifvalfn.p <- function(mu) {
+   log(mu + 0.01)
+ }

> varfn.p <- function(mu) {
+   mu
+ }

> drfn.p <- function(mu) {
+   mu
+ }

> devfn.p <- function(mu, y, event,
+   scale, weights) {
+   sum(weights * (y * log(mu) - mu))
+ }

> scale.updatefn.p <- function(mu, y,
+   event, weights, scale) {
+   1
+ }
```

Note that in order to handle zero counts, a small constant is added to `mu` when taking its logarithm in the function `ifvalfn.p`. When defining these functions, it is important to include an argument in the argument list of a function even when it is not used, e.g. though the `scale.updatefn` returns a constant, all of the arguments – `mu`, `y`, `event`, etc. – must be explicitly included in the function call in the order shown.

To try out `HGglm` we first generate an artificial dataset in which the matrix of explanatory variables is of size  $40 \times 1000$  and where, in the true model, only the intercept and the coefficient of the first explanatory variable are non-zero:

```
> set.seed(95873)
> X <- matrix(runif(40 * 1000, -5, 10),
+   40, 1000)

> Beta <- c(0.1, 0.3)
> Eta <- cbind(rep(1, 40), X[, 1]) %*%
+   Beta
> Mu <- exp(Eta)
> y <- rpois(40, Mu)

> y
```

```
[1] 1 1 9 6 5 0 2 0 1 27 1 3 2
[14] 3 7 13 0 0 14 7 0 17 5 1 12 5
[27] 0 9 16 8 16 0 8 8 19 3 6 0 9
[40] 6
```

First, to fit a sparse Poisson regression to these data using the 'built-in' functionality of `HGglm` we invoke it with the following arguments:

```
> res.HGglm <- HGglm(X, y, model = "P",
+   b0sc = -1)
```

The argument `b0sc` determines the length of the initial estimate of  $\beta$ , and when it is set to a negative number, `HGglm` generates an initial value. The results object is a list containing the following components:

```
> names(res.HGglm)

[1] "beta"  "S"      "fv"     "varids"
[5] "scale" "model"
```

The variables that have been selected are contained in the component `varids`:

```
> res.HGglm$varids

[1] 0 1
```

and we can see that only the intercept and first explanatory variable have been chosen. The estimated coefficients are given by

```
> round(res.HGglm$beta[res.HGglm$varids +
+   1], 2)

[1] 0.12 0.29
```

For this artificial example, the agreement with the true value is pretty good!

Now to fit the same model using the auxiliary functions defined above, we invoke `HGglm` as follows:

```
> res.HGglm.own <- HGglm(X, y, model = "Own",
+   b0sc = -1, fvalfn = fvalfn.p,
+   ifvalfn = ifvalfn.p, varfn = varfn.p,
+   drfn = drfn.p, devfn = devfn.p,
+   scale.updatefn = scale.updatefn.p)
```

The variables chosen and their estimated coefficients are

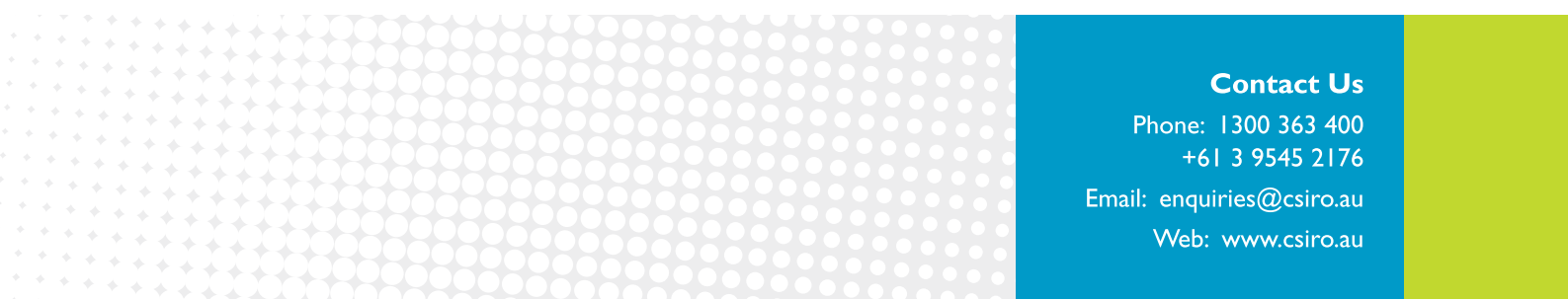
```
> res.HGglm.own$varids

[1] 0 1

> round(res.HGglm.own$beta[res.HGglm$varids +
+   1], 2)

[1] 0.12 0.29
```

and we see that the results are identical to two decimal places.



### Contact Us

Phone: 1300 363 400

+61 3 9545 2176

Email: [enquiries@csiro.au](mailto:enquiries@csiro.au)

Web: [www.csiro.au](http://www.csiro.au)

### Your CSIRO

Australia is founding its future on science and innovation. Its national science agency, CSIRO, is a powerhouse of ideas, technologies and skills for building prosperity, growth, health and sustainability. It serves governments, industries, business and communities across the nation.