R Ocean Production Examples (ROPE)

George N. White III

2017-09-13 Wed

Contents

R Ocean Production Examples (ROPE)

1 Introduction

This document provides examples of the steps used in operational estimation of ocean production from remotely sensed data using nearest-neighbour imputation (Trevor Platt et al. 2008). R (Ihaka and Gentleman 1996) is an open source implementation of the S language (Becker, Chambers, and Wilks 1988) developed at Bell Labs in the 1980's.

The data required are *in situ* observations of biomass profiles and P–I relationships, and ocean colour remote sensing images of chlorophyll, PAR, and SST. A coarse outline of the operational process is:

- **curve fitting** – Assign values to the parameters of models for the relationships between depth and biomass and between light and production.
- **nearest-neighbour imputation** – Assign value for these parameters to each satellite image pixel.
- **production calculation** – Compute daily water-column production using a depth and spectrally resolved numerical model.

Simple examples for each of these processing tasks are provided below. For operational use, however, these steps must be performed for large data sets so batch mode processing is required.

For the purposes of these examples, ROPE will be used under unix-like systems (macOS and linux). ROPE optionally includes a version of the Fortran program `dwcpn` by Platt and Sathyendranath. The R code can be used on Windows, linux, or Mac OS X. The `dwcpn` program can be used on any platform for which a modern Fortran compiler is available.

ROPE exploits the data manipulation and plotting capabilities of the S language to prepare input data for production calculations and analyze the results. The purpose of these examples is to help readers understand basic principles, not to provide a system suitable for large-scale calculations involving millions of pixels.

The document includes the complete source code for ROPE as well as test data sets. The original document is maintained using Emacs Org-mode, which supports Reproducble Research and Literate Programming techniques. The complete sources as well as output from the examples are included in the source document, and the program files can be created directly from the document. This means the document and sources will always agree. The reader must, however, be prepared to skip over the program listings at first reading.

Nonlinear least-squares curve-fitting methods available in R are used to obtain estimates for the values of model parameters needed to calculate daily water-column production with non-uniform biomass, and then to support the use of the resulting estimates to perform production calculations.

Before reading this document users are well advised to work through "An Introduction to R" included in the documentation of the base R package. To get information about a particular function, enter `?function` to see the on-line documentation. There are a number of excellent books that describe the R language as well as an electronic (PDF) journal.

The ROPE package runs on any platform for which the Ocean Production software and R version 3.4 or higher are available. The software has been tested with MacOSX and various linux distributions.

2 Contents of the ROPE package

The ROPE package defines the following R functions:

dwcpn driver for the dwcpn program. This function passes data stored in a data frame to the program and then reads the output and stores it in a new data frame.

Production evaluate the P–I function as a function of irradiance for given parameter values

SGauss evaluate the shifted-Gaussian function as a function of depth for given parameter values

SGplot plot shifted-Gaussian curves

PIplot plot P–I curves

nlsrFitPI estimate parameters of P–I curves from data

nlsrFitSG estimate parameters of biomass–depth curves from data

These functions are discussed below.

To run the examples below you may also want to have the sample data (`.csv`) files. You may find it useful to run R in the `rstudio` GUI environment.

Run `R` and then type the commands to load the package and list the functions (as well as any data objects in the workspace):

```
$ tar xvzf .../ROPE.tar.gz
$ mkdir work-$(date +%F)
$ cd work-$(date +%F)
$ R

[...]
> source('rope.R')
> ls()
```

3 Models

Curve fitting is used to normalize *in situ* data sets that may consist of varying numbers of observations taken at varying levels of the independent variable. By reducing a set of *in situ* observations to a set of values for a small number of parameters, the same production program may be applied for many different conditions.

This section describes the models for biomass profiles and the relationships between production and irradiance.

3.1 Nonuniform biomass model

Here, we are interested in the case where biomass has the typical pattern of a subsurface maximum. This pattern is commonly modelled using a shifted Gaussian model (1), depicted in Figure 1.

$$B(z) = B_0 + \frac{h}{\sigma\sqrt{2\pi}} e^{-\frac{(z - z_m)^2}{2\sigma^2}} \tag{1}$$

Note that the curve in Figure 1 is drawn in the form of a mathematical function rather than a pictorial view with depth increasing downwards. It is important to note that the surface biomass:

$$B(0) = B_0 + \frac{h}{\sigma\sqrt{2\pi}}\,e^{-\dfrac{z_m^2}{2\sigma^2}} \tag{2}$$

depends on all four parameters. The reader should derive a formula to scale a shifted-Gaussian biomass profile (represented by value of the four parameters) so that $B(0)$ matches a given value.

3.2 Production – irradiance model

Here we will consider the model `(2)` for the relationship between production, P, and irradiance, I (Figure 2):

$$P^B(I) = P(I)/B = P_m^B(1 - e^{-\dfrac{\alpha^B I}{P_m^B}}) \tag{3}$$

3.3 Numerical evaluation

This section provides simple numerical examples for the two models discussed above using the S language.

The model equations can be evaluated numerically using the following R function:

```
SGauss <- function(z, B0, h, sigma, zm)
{
   (B0 + h*exp(-(z-zm)^2/(2*sigma*sigma))
/(sigma*sqrt(2*pi)))
}

Production <- function(Irradiance, alpha, PmB)
{
   return ( PmB*(1-exp(-alpha*Irradiance/PmB)))
}
```

The following interactive session was used to generate the values in Figure `1`.

```
> source('rope.R')
> z    <-seq(0,50,by=0.2)
> B    <-SGauss(z, B0=0.2, h=15, sigma=4, zm=25)
> plot(B~z,type='l',ylim=c(0,2))
```

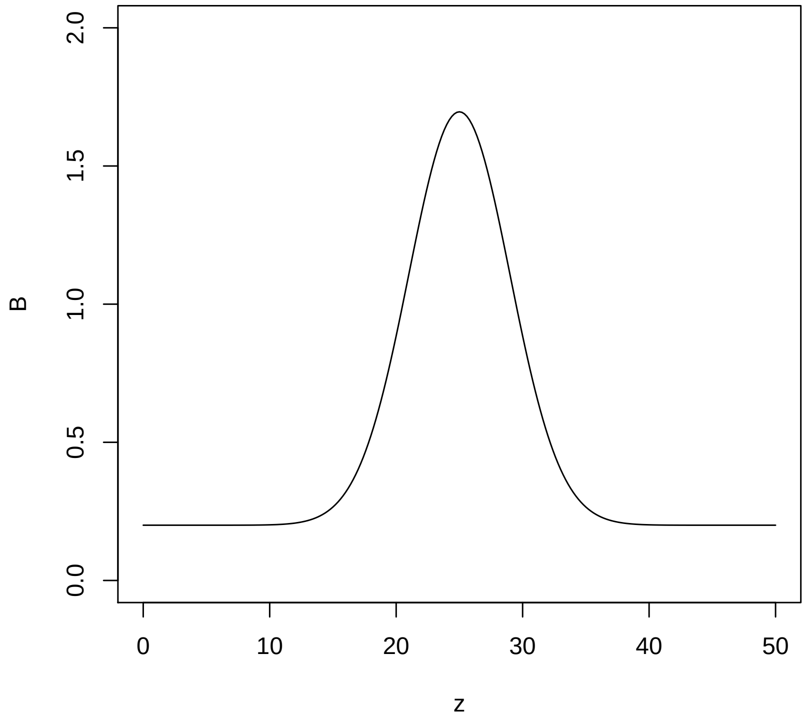

Figure 1: Shifted Gaussian model.

Note first the use of the `c()` function to create the parameter vector with named components and also the representation, or model, of the relationship between biomass and depth as `B~z`.

The commands below were used to create a PDF version of the plot.

```
> pdf('B.pdf')
> plot(B~z, type='l')
> dev.off()
```

Similar steps are used to create Figure 2.

```
> I <-seq(0,2.0,by=0.05)
> P<-Production(I, alpha=0.8, PmB=0.2)
> plot(P~I, type='l')
```

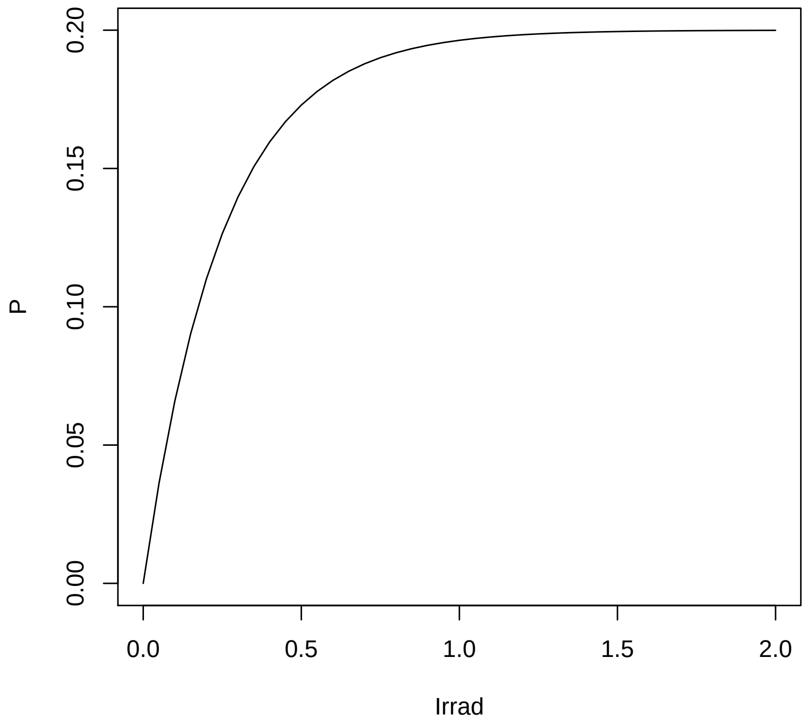

Figure 2: Production – Irradiance model.

The plot can be saved to a PDF file as follows:

```
> pdf('P.pdf')
> plot(P~I, type='l')
> dev.off()
```

4 Curve fitting

This section provides low-level examples of curve fitting using R.

The fits are obtained using the nonlinear least-squares estimation tools in the `nlsr` library – do not use `library(nls)`, which may be suggested in older documentation, as `nlsr` is intended to replace `nls` in the future.

4.1 Biomass profiles

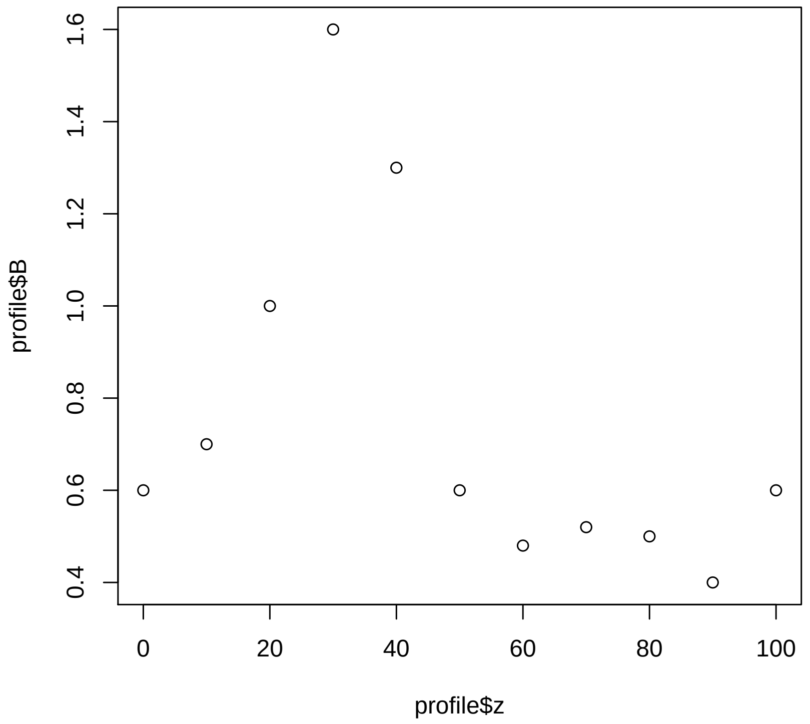

Figure 3: Sample biomass profile data set.

The following example data set is shown in Figure 3.

```
profile.df <- data.frame(z=seq(0, 100, by=10),
     B=c(0.6, 0.7, 1.0, 1.6, 1.3, 0.6, 0.48, 0.52, 0.50, 0.4, 0.6))
```

Given a data file, `profile.csv`, the data can be loaded into the R workspace using `read.table()` as follows:

```
> profile.df <- read.table('profile.csv', header=T, sep=',')
> str(profile.df)
`data.frame': 11 obs. of  2 variables:
 $ z: int  0 10 20 30 40 50 60 70 80 90 ...
 $ B: num  0.6 0.7 1 1.6 1.3 0.6 0.48 0.52 0.5 0.4 ...
> edit(profile.df)
> with(profile.df, plot(B~z))
```

Note the use of the `with()` function. This runs the `plot(..)` command after redefining the values of `B` and `z` from the `profile.df` data frame. If you want to use your on data, you need to set the variable names to match the example or edit the functions to match the names in your data.

The above plot shows biomass as a function of depth. It may also be useful to exchange the axes:

```
with(profile.df, plot(z~B, ylim=c(100,0)))
```

The user can create new data files using a spreadsheet to write data in the `.csv` file format, or can use R's own data manipulation tools to create data sets directly.

It is often useful to plot the data set and model curve together. The following function provides this capability. By setting a default of `NA` for the parameters, the function can be used to plot only data points without the curve.

```
SGplot<-function(data, B0=NA, h=NA, sigma=NA, zm=NA){
plot.sg<-function(z, B, B0, h, sigma, zm ){
  plot(z~B,ylim=c(max(z),0))
  lines(seq(0,max(z))~SGauss(seq(0,max(z)), B0, h, sigma, zm))
  }
with(data,plot.sg(z,B, B0, h, sigma, zm))
}
```

Experience fitting the shifted-Gaussian model to 1000's of depth profiles has shown that the default parameterization does not always perform well. In some cases the model shold be simplified by eliminating either the background biomass (*e.g.*, $B_0 = 0$) or the Gaussian peak (*e.g.*, $h = 0$). In some cases the biomass profile decreases with depth, corresponding to a Gaussian model with the peak above the surface, (*e.g.*, $z_m < 0$ in a coordinate system with depth increasing downwards). Using the original model parameterization, the fitting procedure may generate a large value for B_0, $z_m > 0$, and a negative height, $h < 0$. This result is physically unrealistic, as it implies the highest biomass at great depths. To exclude such behaviour, we modify the model by introducing new parameters, a, b, and c, defined by $B_0 = \exp(a)$,

$h = \exp(b)$, and $\sigma = \exp(c)$ (in the programs we use $a =$ `lnB0`, $b =$ `lnh`, and $c =$ `lns` for clarity).

```
require(nlsr)
m <- B ~ B0 + h*exp(-(z-zm)^2/(2*sigma*sigma))
        /(sigma*sqrt(2*pi))
s <-  c(B0=0.5, h=26, sigma=9, zm=32)
profile.nlsr <- nlxb( m, data=profile.df, start = s)
```

Printing `profile.nlsr` provides a brief description of the results:

```
> profile.nlsr
nlsr object: x
residual sumsquares =  0.049345  on  11 observations
    after  11    Jacobian and  12 function evaluations
  name       coeff     SE       tstat    pval      gradient    JSingval
B0          0.520555  0.03396   15.33  1.213e-06  -4.441e-16    3.317
h          25.7394    2.491     10.33  1.725e-05  -1.069e-15    0.1383
sigma       9.24957   0.8584    10.78  1.305e-05   -5.24e-16    0.1085
zm         31.6538    0.7741    40.89  1.364e-09   4.358e-16    0.03274
```

The `summary()` function provides more detail:

```
> summary(profile.nlsr)
$residuals
 [1] -0.07626620 -0.10778224  0.02252864  0.01311365
 [5] -0.04054490  0.07583073  0.05069394  0.00076056
 [9]  0.02055614  0.12055484 -0.07944516
attr(,"gradient")
      B0          h       sigma          zm
 [1,]  1 1.2351e-04  3.6814e-03 -1.1762e-03
 [2,]  1 2.7842e-03  3.4714e-02 -1.8138e-02
 [3,]  1 1.9502e-02  3.1880e-02 -6.8376e-02
 [4,]  1 4.2447e-02 -1.1434e-01 -2.1120e-02
 [5,]  1 2.8707e-02 -1.4843e-02  7.2082e-02
 [6,]  1 6.0326e-03  4.9256e-02  3.3297e-02
 [7,]  1 3.9391e-04  9.1988e-03  3.3593e-03
 [8,]  1 7.9923e-06  3.6001e-04  9.2204e-05
 [9,]  1 5.0388e-08  3.6905e-06  7.3289e-07
[10,]  1 9.8708e-11  1.0655e-08  1.7327e-09
[11,]  1 6.0083e-14  8.9617e-12  1.2354e-12
```

```
$sigma
[1] 0.08396

$df
[1] 4 7

$cov.unscaled
             B0        h    sigma        zm
B0     0.163641  -8.0024  -1.8891  0.025952
h     -8.002434 880.4754 207.9024 -2.874374
sigma -1.889108 207.9024 104.5321 -1.631644
zm     0.025952  -2.8744  -1.6316 85.004632

$param
      Estimate Std. Error t value   Pr(>|t|)
B0     0.52055   0.033964  15.327 1.2133e-06
h     25.73941   2.491336  10.332 1.7248e-05
sigma  9.24957   0.858417  10.775 1.3052e-05
zm    31.65383   0.774096  40.891 1.3637e-09

$resname
[1] "profile.nlsr"

$ssquares
[1] 0.049345

$nobs
[1] 11

$ct
[1] " " " " " " " "

$mt
[1] " " " " " " " "

$Sd
[1] 3.316762 0.138257 0.108455 0.032743

$gr
```

```
              [,1]
B0    -4.4409e-16
h     -1.0686e-15
sigma -5.2396e-16
zm     4.3582e-16

$jeval
[1] 11

$feval
[1] 12

attr(,"pkgname")
[1] "nlsr"
attr(,"class")
[1] "summary.nlsr"
```

The `coef()` function returns the estimated parameter values:

```
> pars <- coef(profile.nlsr)
      B0        h    sigma       zm
 0.52055 25.73941  9.24957 31.65383
attr(,"pkgname")
[1] "nlsr"
> SGplot(profile.df, pars)

ef<-function(z,zm,lns)
{ # lns=log(sigma)
  ((z-zm)*(z-zm)/(2*exp(2*lns)))
}
profile.nls <- nls(
    B ~ exp(lnB0) + exp(lnh-ef(z,zm,lns) - lns - 0.9184),
    profile.df,
    start = c(lnB0=log(0.5), lnh=log(26), lns=log(9), zm=32),
    trace = TRUE)

0.05593941 :  -0.6931472  3.2580965  2.1972246 32.0000000
0.04942556 :  -0.6490373  3.2411933  2.2160070 31.7112855
0.04934863 :  -0.6523228  3.2463127  2.2229820 31.6676586
0.0493454 :  -0.6527399  3.2472178  2.2242053 31.6563992
```

```
0.04934526 :  -0.6528369  3.2474330  2.2245050 31.6544285
0.04934525 :  -0.6528548  3.2474730  2.2245610 31.6539497
0.04934525 :  -0.652859  3.247482  2.224574 31.653860
0.04934525 :  -0.6528598  3.2474841  2.2245765 31.6538393

> profile.nls
Nonlinear regression model
  model:  B ~ exp(lnB0) + exp(lnh - ef(z, zm, lns) - lns - 0.9184)
   data:  profile.df
   lnB0     lnh     lns      zm
-0.6529  3.2475  2.2246 31.6538
 residual sum-of-squares: 0.04935

Number of iterations to convergence: 7
Achieved convergence tolerance: 3.009e-06

> summary(profile.nls)

Formula: B ~ exp(lnB0) + exp(lnh - ef(z, zm, lns) - lns - 0.9184)

Parameters:
     Estimate Std. Error t value Pr(>|t|)
lnB0 -0.65286    0.06525  -10.01 2.13e-05 ***
lnh   3.24748    0.09679   33.55 5.41e-09 ***
lns   2.22458    0.09281   23.97 5.59e-08 ***
zm   31.65384    0.77410   40.89 1.36e-09 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.08396 on 7 degrees of freedom

Number of iterations to convergence: 7
Achieved convergence tolerance: 3.009e-06
```

The log-transformed coefficients can be converted to their untransformed values:

```
> pars <- coef(profile.nls)
> pars[1:3]<- exp(pars[1:3])
> names(pars)= c('B0', 'h', 'sigma', 'zm')
> pars
```

```
       B0          h     sigma         zm
 0.520555 25.725536  9.249565 31.653839
> with(pars, SGplot(profile.df, B0, h, sigma, zm)
```

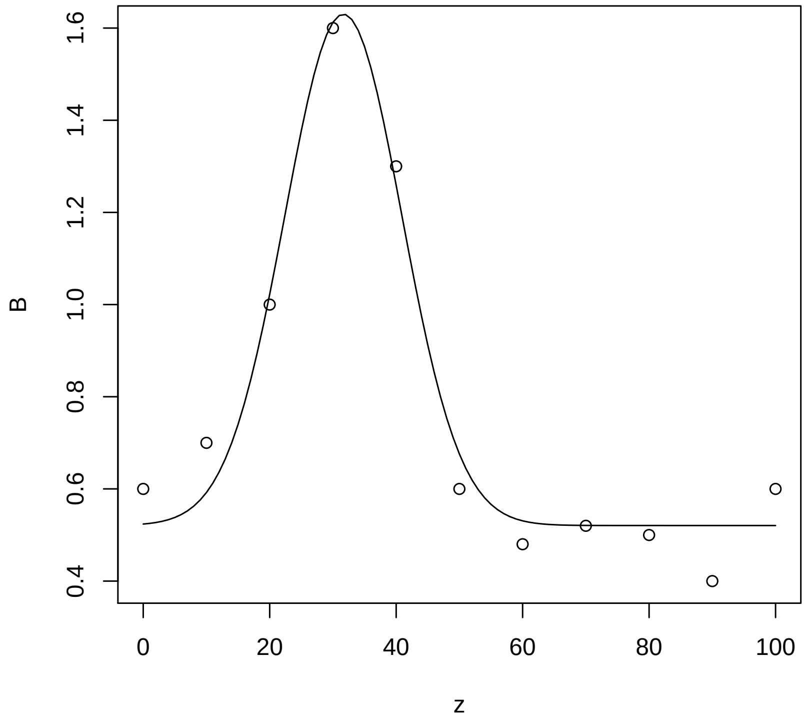

Figure 4: Sample biomass profile data set with curve fit using log-transformed parameters.

The result is shown in Figure 4:

```
> pdf('lnfitB.pdf')
> with(profile.df,plot(B~z))
> lines(SGauss(seq(0,100), B0=exp(-0.65285980), h=exp(3.2474841),
        sigma=exp(2.2245765), zm=31.6538393)~seq(0,100))
> dev.off()
```

4.2 Production – Irradiance

Here we use R to estimate parameter values for models of the relationship between production and irradiance described by T. Platt, Gallegos, and Harrison 1980.

Consider the following sample data set, from the file `p-i.csv`:

```
x <- c(
536.8, 2.80,
408.5, 2.51,
356.3, 2.31,
313.5, 2.55,
161.5, 2.43,
133.0, 2.14,
111.2, 2.10,
98.8, 1.95,
89.3, 1.71,
79.3, 1.47,
53.2, 1.36,
42.8, 1.06,
38.0, 0.98,
34.2, 0.88,
30.9, 0.68,
27.6, 0.58,
19.0, 0.46,
15.7, 0.38,
13.8, 0.32,
12.2, 0.21,
11.1, 0.16,
9.9, 0.14,
6.6, 0.10,
5.6, 0.07,
4.9, 0.05,
2.2, 0.02,
1.9, 0.01)
dim(x) <- c(2, 27)
PI.df <- as.data.frame(t(x))
names(PI.df) <-  c("I", "P")
```

Figure 5 provides a plot of the sample data set.

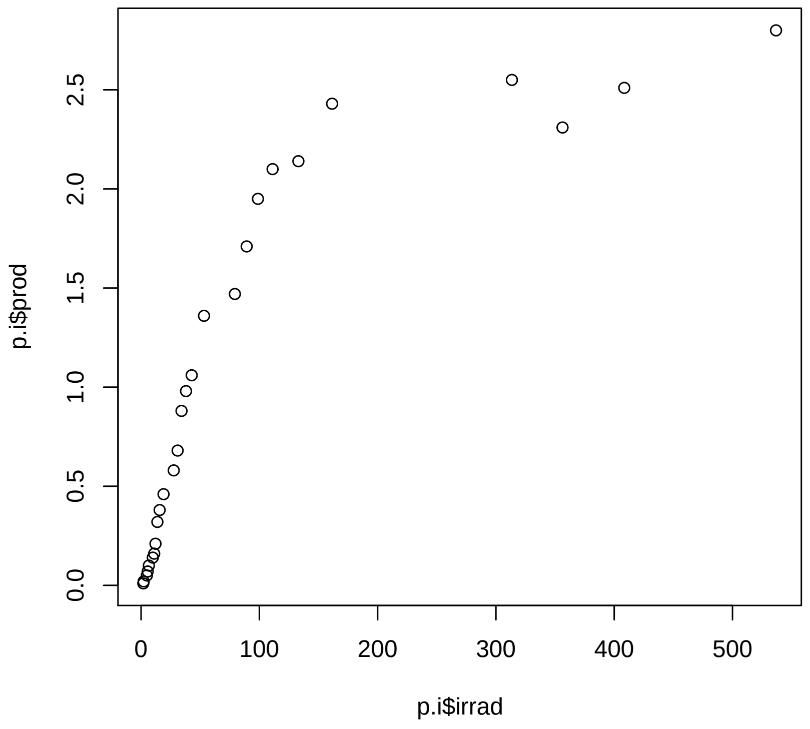

Figure 5: Sample production and irradiance data set.

```
> pi.df<-read.table('p-i.csv', header=T, sep=',')
> str(pi.df)
> with(pi.df, plot(P~I))
> PIplot(pi.df,NA)
```

Again, the parameters can only assume positive values. Experience has shown that the confidence intervals for the rate constant, $rca = \alpha/P_m^B$, may have negative lower endpoints, so appropriate to introduce a log-transformed variable: $lrca = \log(\alpha/P_m^B)$.

```
pi.nlsr <- nlxb( P ~ PmB*(1-exp(-exp(lrca)*I)),
                data=PI.df,
                start = c(PmB=3.0,lrca=-3.5))
```

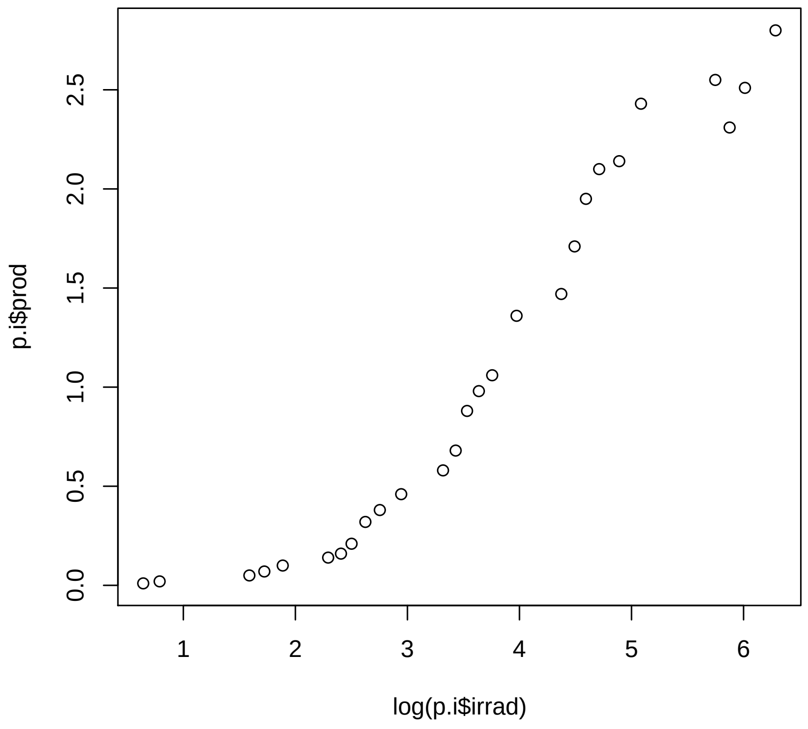

Figure 6: Sample production and irradiance data set, log-transformed irradiance.

```
vn:[1] "P"    "PmB"  "lrca" "I"
no weights
> pi.nlsr
nlsr object: x
residual sumsquares =  0.40098  on  27 observations
    after  8    Jacobian and  9 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
PmB              2.63512       0.06688       39.4  5.024e-24  -
1.912e-13       3.878
lrca            -4.44148       0.06208     -71.54  1.922e-30   1.684e-
13       1.486
```

```
> pars <- coef(pi.nlsr)
> pars["alpha"] <- exp(lrca)*pars["PmB"]
> names(pars)<- c('PmB','lrca', 'alpha')
> pars
      PmB      lrca     alpha
 2.635119 -4.441478  0.031038
attr(,"pkgname")
[1] "nlsr"
> PIplot(PI.df, pars["alpha"], pars["PmB"])
```

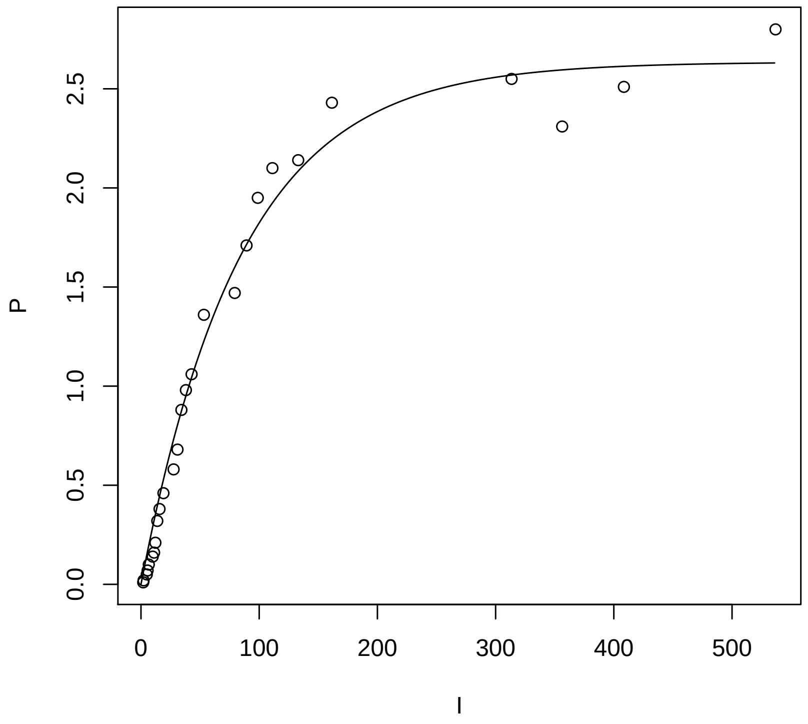

Figure 7: Sample production–irradiance data set with curve fit using log-transformed parameters.

#> pdf('lnfitP.pdf') #> PIplot(pi.df,pars) #> dev.off()

As before, it is helpful to define a function to plot the data together with the model curve:

```
PIplot<-function(data, alpha, PmB){
plot.pi<-function(P,I, alpha, PmB){
  main<-sprintf("Production: alpha=%f, PmB=%f", alpha, PmB)
  plot(P~I, main=main)
  lines(Production(seq(0,max(I)), alpha, PmB)~seq(0,max(I)))
}
with(data,plot.pi(P,I, alpha, PmB))
}
```

5 Nearest neighbour imputation

Imputation refers to the class of statistical methods that assign values to parameters whose values have not been directly observed based on *reference* measurements. In remote sensing imputation is used to infer values for variables, y, that cannot be measured remotely in terms of the remotely sensed variables, x.

The R package `yaImpute` implements a collection of methods based on averages of the reference measurements that are most like the x value for each case where y cannot be determined directly. In typical remote sensing applications, the reference measurements are based on *ground truth* measurements of a localized sample, while remote measurements represent averages over much large spatial extents. In this case, averaging reference values will tend to produce estimates that better resemble the remote sensing data in terms of spatial scales and statistical variability than do individual reference observations.

You do not need to have the ROPE functions loaded in order to use the `yaImpute` library.

5.1 Files

The examples below were based on the following files for an area in the Northwest Atlantic:

```
> list.files()
2004maya-PP_TC.asc
2004maya-cfr_TC.asc
2004maya-chl_TC.asc -- SeaWiFS chlorophyll
```

```
2004maya-nav_TC.asc
2004maya-par_TC.asc
2004maya-sst_TC.asc -- AVHRR SST
cp-20051114.csv -- chlorophyll profile parameters
pi-20051115.csv -- photosynthesis--irradiance parameters
```

In order to perform productioon calculations you will need to provide data for your region of interest.

To load the data and convert missing value codes to `NA`, enter the following commands:

```
> chl=read.table("2004maya-chl_TC.asc",header=T)
> sst=read.table("2004maya-sst_TC.asc",header=T)
> chl$chl_oc4[chl$chl_oc4 < 0.01] = NA
> str(chl)
'data.frame':   243264 obs. of  1 variable:
 $ chl_oc4: num   4.47 17.87  6.63  2.77  3.27 ...
> sst$sst[sst$sst < (-2.0)]=NA
> str(sst)
'data.frame':   243264 obs. of  1 variable:
 $ sst: num  0.125 0 0.125 NA NA NA NA 0.125 0.25
             0.125 ...
```

5.2 Reference data

Before applying imputation to satellite data, it is useful to check the performance on the reference data sets. In the example below, all the data are used. You may want to experiment using small subsets based on proximity in time and space to your region of interest.

5.2.1 Chlorophyll profile data

```
> require(yaImpute)
> cp.20051114=read.table('cp-20051114.csv', sep=',', header=T)
> str(cp.20051114)
'data.frame':   1584 obs. of  16 variables:
 $ lat    : num  42.8 42.7 42.6 42.7 42.8 ...
 $ lon    : num  -64 -64 -64 -64 -64 ...
 $ dn     : int  181 182 182 182 182 183 183 183 183 183 ...
 $ day    : int  29 30 30 30 30 1 1 1 1 1 ...
 $ mon    : int  6 6 6 6 6 7 7 7 7 7 ...
 $ year   : int  1976 1976 1976 1976 1976 1976 1976 1976 1976 1976 ...
 $ h      : num  28.7 27.6 14.4 34.1 24.7 ...
 $ sigma  : num  31.38 12.82  7.84 20.12  8.48 ...
 $ zm     : num  35.4 45.6 32.2 27.4 19.8 ...
 $ Bo     : num  0.011 0.126 0.077 0 0.047 0.14 0.13 0.101 0.043
                 0.088 ...
 $ Rho    : num  0.97 0.87 0.91 1 0.96 0.93 0.89 0.93 0.89 0.94 ...
 $ dataset: Factor w/ 68 levels "2000-002","2000-009",..:
       57 57 57 57 57 57 57 57 57 57 ...
 $ stn    : Factor w/ 548 levels "-99","1","10",..:
       206 41 46 54 62 82 90 97 106 111 ...
 $ chlsur : num  0.204 0.128 0.077 0.268 0.123 0.279 0.13 0.106
                 0.308 0.088 ...
 $ tempsur: num  14 14.4 -9.9 -9.9 14 -9.9 -9.9 14 13.8 -9.9 ...
 $ tz     : num  0 10 -9.9 -9.9 10 -9.9 -9.9 10 10 -9.9 ...
```

The x-variables will temperature and log-transformed chlorophyll:

```
> cp.20051114$logchl0=log(cp.20051114$chlsur)
> x=cp.20051114[,c(15,17)]
> str(x)
'data.frame':   1584 obs. of  2 variables:
 $ tempsur: num  14 14.4 -9.9 -9.9 14 -9.9 -9.9 14 13.8 -9.9 ...
 $ logchl0: num  -1.59 -2.06 -2.56 -1.32 -2.10 ...
```

The missing value codes are replaced with `NA`:

```
> x$tempsur[x$tempsur==(-9.9)]=NA
> str(x)
'data.frame':   1584 obs. of  2 variables:
 $ tempsur: num  14 14.4 NA NA 14 NA NA 14 13.8 NA ...
 $ logchl0: num  -1.59 -2.06 -2.56 -1.32 -2.10 ...
```

The y variables will be $\log(1+h)$, $\log(\sigma)$, z_m, and $\log(1+B_0)$ (h and B_0 can have values of zero, so we can't use $\log(h)$ or $\log(B_0)$).

```
> y=cp.20051114[,7:10]
> y[,c(1,2,4)]=log(1+y[,c(1,2,4)])
> colnames(y)=c('ln1ph','ln1psigma','zm','ln1pB0')
```

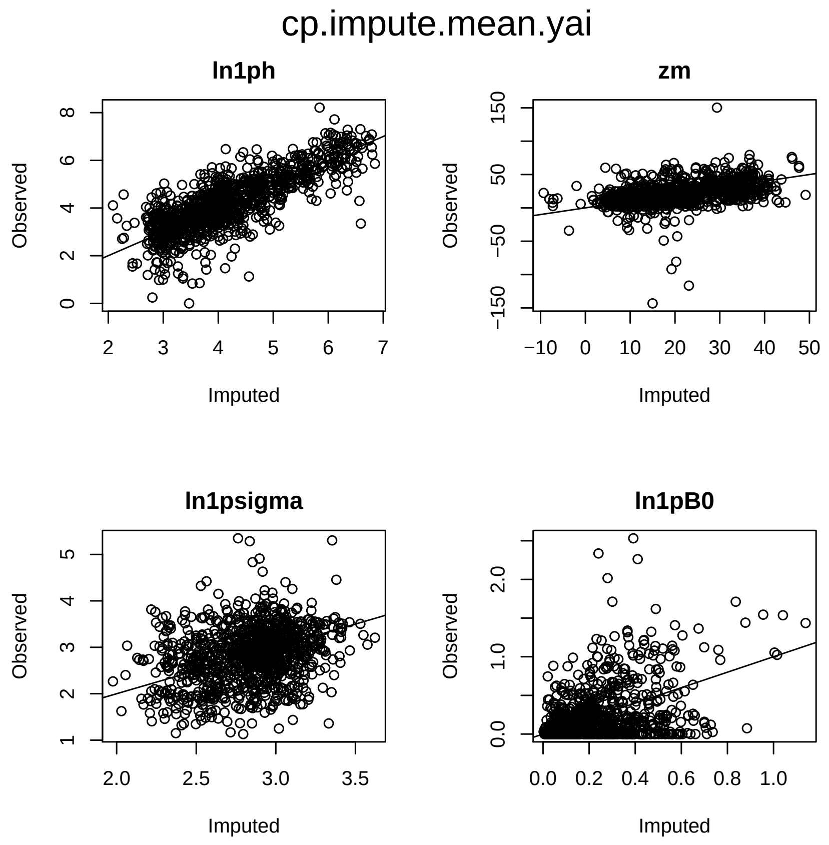

Figure 8: Nearest neighbour imputation for profile reference data using Mahalanobis distances with means for $k = 10$.

We use `yai` with Mahalanobis distances and $k = 10$, then compare nearest-neighbor with the means of 10 nearest neighbors (Figure 7). Note that the `impute` function may take a long time to run on slower computers.

```
> cp.mal.10.yai <- yai(x=x, y=y, method="mahalanobis",k=10)
> cp.impute.mean.yai=impute(cp.mal.10.yai,method='mean')
> compare.yai(cp.mal.10.yai,cp.impute.mean.yai)
            cp.mal.10.yai.rmsdS cp.impute.mean.yai.rmsdS
ln1ph                 0.8311586                0.6093150
ln1psigma             1.3048093                0.9683073
zm                    1.3019823                0.9089843
ln1pB0                1.1908467                0.9055872
```

```
> plot(cp.impute.mean.yai)
```

```
> pdf('cpyai.pdf')
> plot(cp.impute.mean.yai)
> dev.off()
```

5.2.2 *P–I* data

The application to *P–I* data follows the same outline as was used above for the profile data.

```
> pi.20051115=read.table('pi-20051115.csv', sep=',', header=T)
> str(pi.20051115)
'data.frame':   1638 obs. of  12 variables:
 $ lat   : num  44.5 44.5 44.5 47.0 47.0 ...
 $ lon   : num  -57.0 -57.0 -57.0 -52.5 -52.5 ...
 $ day   : int  15 15 15 16 16 16 17 17 17 18 ...
 $ mon   : int  5 5 5 5 5 5 5 5 5 5 ...
 $ year  : int  1996 1996 1996 1996 1996 1996 1996 1996 1996 1996 ...
 $ pm    : num  0.74 0.47 1.36 0.65 1.17 1.28 0.61 1.36 1.72 2.13 ...
 $ alpha : num  0.028 0.017 0.032 0.032 0.031 0.038 0.029 0.023
                0.021 0.068 ...
 $ z     : num  50 30 10 50 30 10 50 30 10 50 ...
 $ temp  : num  1.8 2.7 3.5 -0.2 1.4 1.4 -0.6 0.8 0.8 -1.4 ...
 $ chl   : num  4.29 2.94 0.9 2.52 0.49 0.46 1.04 0.48 0.44 7.91 ...
 $ id    : Factor w/ 1430 levels "001201  ","001202  ",..: 488 489
           490 491 492 493 494 495 496 497 ...
 $ cruise: Factor w/ 56 levels "2000-002  ","2000-009  ",..: 36 36 36
           36 36 36 36 36 36 ...
```

As in the previous example, we define x and y for the reference sample. Here we do not need to use log(1+) transformations as there are no zero values.

```
> x=pi.20051115[,9:10]
> x[,2]=log(x[,2])
> colnames(x)=c("temp","logchl" )
> str(x)
'data.frame':   1638 obs. of  2 variables:
 $ temp  : num  1.8 2.7 3.5 -0.2 1.4 1.4 -0.6 0.8 0.8 -1.4 ...
 $ logchl: num   1.456  1.078 -0.105  0.924 -0.713 ...
```

```
> y=log(pi.20051115[,6:7])
> colnames(y)=c("logpm", "logalpha")
> str(y)
'data.frame':   1638 obs. of  2 variables:
 $ logpm   : num  -0.301 -0.755  0.307 -0.431  0.157 ...
 $ logalpha: num  -3.58 -4.07 -3.44 -3.44 -3.47 ...
```

Finally, we perform the imputation:

```
> pi.mal.10.yai=yai(x=x, y=y, method="mahalanobis",k=10)
> pi.impute.mean.yai=impute(pi.mal.10.yai,method='mean')
> plot(pi.impute.mean.yai)
> compare.yai(pi.mal.10.yai,pi.impute.mean.yai)
         pi.mal.10.yai.rmsdS pi.impute.mean.yai.rmsdS
logpm               1.036501                0.8329870
logalpha            1.117363                0.8624002
```

#+NAME fig-piyai
#%> pdf('piyai.pdf') #%> plot(pi.impute.mean.yai) #%> dev.off()

5.3 Imputations for remote sensing images

Here we use imputation to generate values for P_m^B and α for the first half of May, 2004 in the region defined for this course.

```
> pi.20051115=read.table('pi-20051115.csv', sep=',', header=T)
> dim(pi.20051115)
[1] 1638   12
```

Since we are interested in spring conditions, we will use data for April, May, and June. This gives 827 reference observations.

```
pimay=pi.20051115[(pi.20051115$mon > 3)&(pi.20051115$mon < 7),]
> dim(pimay)
[1] 827  12
```

The variables x and y are defined as before for the reference data:

```
x=pimay[,9:10]
x[,2]=log(x[,2])
colnames(x)=c('temp','logchl')
y=log(pimay[,6:7])
colnames(y)=c('logpm','logalpha')
```

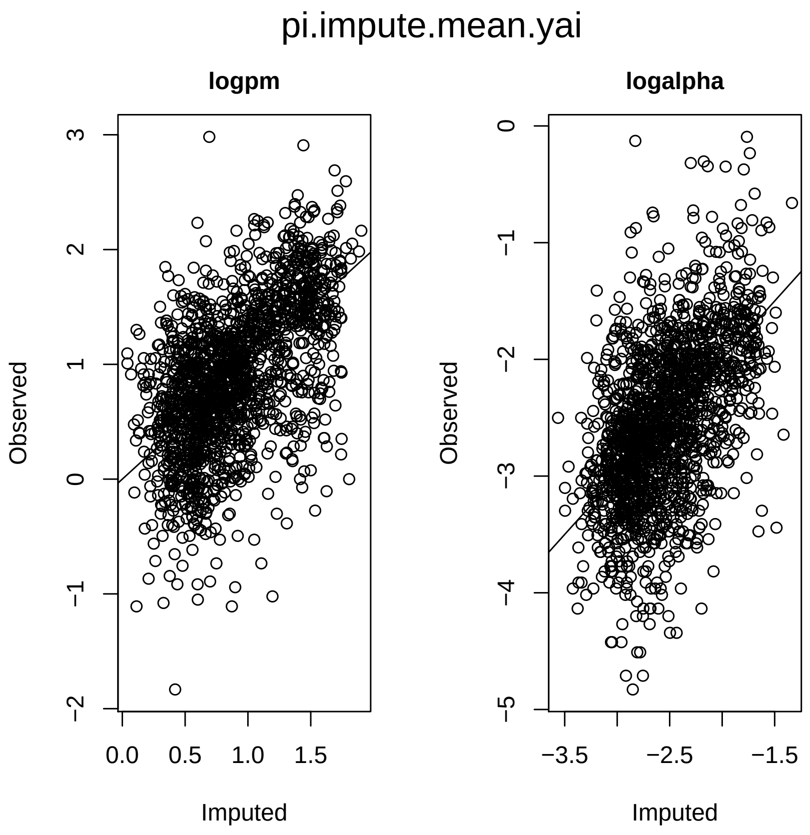

Figure 9: Nearest neighbour imputation for *P–I* reference data using Mahalanobis distances with means for $k = 10$.

Next we add some remotely sensed values to the x variable, using the SST and chlorophyll data frames created earlier:

```
> xrs=sst
> names(xrs)='temp'
> xrs$logchl=log(chl[,1])
> str(xrs)
'data.frame':   243264 obs. of  2 variables:
 $ temp  : num  0.125 0 0.125 NA NA NA NA 0.125 0.25 0.125 ...
 $ logchl: num  1.50 2.88 1.89 1.02 1.18 ...
> xp=rbind(x,xrs[1:10,])
> str(xp)
```

```
'data.frame':   837 obs. of  2 variables:
 $ temp  : num  1.8 2.7 3.5 -0.2 1.4 1.4 -0.6 0.8 0.8 -1.4 ...
 $ logchl: num   1.456  1.078 -0.105  0.924 -0.713 ...
```

Finally, we perform the imputation as before, replacing `x` by `xp`. Note that the remote sensing observations with `NA` values are removed from the analysis:

```
> pirs.mal.10.yai=yai(x=xp, y=y, method="mahalanobis",k=10)
Warning in yai(x = xp, y = y, method = "mahalanobis", k = 10) :
         4 x observation(s) removed
> pirs.impute.mean.yai=impute(pirs.mal.10.yai,method='mean')
```

To display the imputed values, we list the final 10 records. Note the two columns labelled with `.o`, indicating that there are no reference observations for these 10 records:

```
> pirs.impute.mean.yai[828:837,]
         logpm  logalpha logpm.o logalpha.o
828  0.6853688 -3.099658      NA         NA
829  0.4392359 -3.030800      NA         NA
830  0.6368739 -2.760982      NA         NA
835  0.5703864 -2.423295      NA         NA
836  0.3604744 -3.006743      NA         NA
837  0.4290576 -2.959832      NA         NA
NA          NA        NA      NA         NA
NA.1        NA        NA      NA         NA
NA.2        NA        NA      NA         NA
NA.3        NA        NA      NA         NA
```

5.3.1 Biomass profile imputation

A similar procedure may be used to impute values for the profile parameters. Note, however, that the biomass profile will ultimately need to be scaled so the surface value $B(0)$ agrees with the satellite value for each pixel:

$$B_{\text{sat}} \approx B(0) = k(\tilde{B}_0 + \frac{\tilde{h}}{\tilde{\sigma}\sqrt{2\pi}} e^{-\frac{\tilde{z}_m^2}{2\tilde{\sigma}^2}}) \tag{4}$$

where k is a scale factor that must be determined for each pixel and $\tilde{B}_0$, $\tilde{h}$, $\tilde{\sigma}$, and $\tilde{z}_m$ are the imputed values derived from *in situ* data. The production calculations will then use $B_0 = k \times \tilde{B}_0$, $h = k \times \tilde{h}$, $\sigma = \tilde{\sigma}$, and $z_m = \tilde{z}_m$.

6 Production calculations

The production calculations are performed using an external program, `dwcpn`. This program was written in Fortran, and can be used independently of R. The program reads two input files, `sam_penguin_globpars.inp` (used to initialize arrays in the program), and the user data file. Two output files are created: a log file and a data file. The input and output files use a cumbersome format that originates in the days when data files were stored on punched cards. This program is suitable for use with small experimental data sets with a few dozen observations as well as larger data sets representing millions of image pixels.

To check that `dwcpn` available, use the `Sys.which()` function in R. On a linux system with `dwcpn` installed to `/usr/local/bin`, you should see:

```
> Sys.which('dwcpn')
                 dwcpn
"/usr/local/bin/dwcpn"
```

In order to take advantage of the capabilities of the S language, small data sets can be stored in the form of data frames. R functions are used to write a data frame to a file in the required format, run the `dwcpn` program, and then read the resulting output data file and return the results in the form of a data frame:

```
dwcpn<-function(data, basename, ...){
local({
  write_dwcpn<-function(data, fnm){
     con <- file(fnm,"w")  # open output file connection
     header<-paste('depth Lat Lon Day alphaB P_mB z_m B_0',
   'h sigma Cloud Yelsub',sep=' ')
     # Fortran-style format
     fmt<-paste('(f7.1,x,f6.2,x,f7.2,x,i3,x,f6.4,x,f6.2,x',
  'f7.2,x,f8.4,x,f6.2,x,f7.2,x,f6.2,x,f4.2)', sep=',')
     cat(file=con, header, fmt, sep='\n')
     # the same format translated to sprintf form
     fmt<-paste('%7.1f %6.2f %7.2f %3i %6.4f %6.2f %7.2f',
'%8.4f %6.2f %7.2f %6.2f %4.2f', sep=' ')
     cat(file=con, sprintf(fmt,data$depth,data$Lat,
 data$Lon,data$Day,data$alphaB,data$P_mB,
 data$z_m,data$B_0,data$h,data$sigma,
 data$Cloud,data$Yelsub),sep='\n')
```

```
     close(con)
  }

  run_dwcpn<-function(fnm, ophome='/SGSP/OP'){
     cmd<-paste("dwcpn", " ", fnm, sep='')
     Sys.setenv(OPHOME=ophome)
     system(cmd)
  }

  read_dwcpn<-function(fnm)  {
     # reformat file
     names<-c("Lat","Lon","Day","h","sigma","z_m","B_0",
      "P_mB","alphaB", "P_ZT", "zp", "Cloud",
      "Yelsub", "status")
     return(read.table(fnm,col.names=names,skip=2))
  }

    fnm<-sprintf('%s.dat', basename)
    write_dwcpn(data,fnm)
    run_dwcpn(fnm)
    fnm<-sprintf('%s.out', basename)
    read_dwcpn(fnm)
})
}
```

If you have restarted R, you may need to load the ROPE functions again, with

```
> source('rope.R')
```

The following is a small input data set provided in the ROPE directory. Copy the file `wcpin.csv` to your working directory or paste the text into a new file.

```
depth,Lat,Lon,Day,alphaB,P_mB,z_m,B_0,h,sigma,Cloud,Yelsub
197,43,-68.5, 15,.123,2.18,16.00,.3350, 75.07,26.97,73.2,.3
197,43,-68.5, 46,.123,2.18,21.29,.3193, 90.04,27.46,70.8,.3
197,43,-68.5, 74,.094,2.58,10.16,.1979, 40.70,24.99,65.3,.3
197,43,-68.5,105,.094,2.58,10.68,.2481, 30.88,23.94,65.5,.3
197,43,-68.5,135,.094,2.58, 6.80,.1611,107.13,17.48,61.8,.3
197,43,-68.5,166,.101,3.56,17.08,.3338, 99.92,13.82,58.6,.3
```

```
197,43,-68.5,196,.101,3.56,19.80,.4389, 54.05,12.29,63.1,.3
197,43,-68.5,227,.101,3.56,18.94,.4314, 44.97, 9.66,58.9,.3
197,43,-68.5,258,.059,2.52,21.25,.1457, 20.46, 6.02,48.6,.3
197,43,-68.5,288,.059,2.52,11.05,.2330, 93.63,18.39,66.9,.3
197,43,-68.5,319,.059,2.52, 9.90,.0359, 70.92,16.89,70.0,.3
197,43,-68.5,349,.123,2.18,10.70,.3508, 60.11,26.47,73.6,.3
```

The following commands run the `dwcpn` program using these data:

```
> wcpin.df<-read.table('wcpin.csv', header=T, sep=',')
> str(wcpin.df)
'data.frame':   12 obs. of  12 variables:
 $ depth : num  197 197 197 197 197 197 197 197 197 197 ...
 $ Lat   : num  43 43 43 43 43 43 43 43 43 43 ...
 $ Lon   : num  -68.5 -68.5 -68.5 -68.5 -68.5 -68.5 -68.5
                -68.5 -68.5 -68.5 ...
 $ Day   : int  15 46 74 105 135 166 196 227 258 288 ...
 $ alphaB: num  0.123 0.123 0.094 0.094 0.094 0.101 0.101
                0.101 0.059 0.059 ...
 $ P_mB  : num  2.18 2.18 2.58 2.58 2.58 3.56 3.56 3.56 2.52 2.52 ...
 $ z_m   : num  16.0 21.3 10.2 10.7  6.8 ...
 $ B_0   : num  0.335 0.319 0.198 0.248 0.161 ...
 $ h     : num   75.1  90.0  40.7  30.9 107.1 ...
 $ sigma : num  27.0 27.5 25.0 23.9 17.5 ...
 $ Cloud : num  73.2 70.8 65.3 65.5 61.8 58.6 63.1 58.9 48.6 66.9 ...
 $ Yelsub: num  0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 ...
> wcpout.df<-dwcpn(wcpin.df,'test')
```

The output is:

```
> str(wcpout.df)
'data.frame':   12 obs. of  14 variables:
 $ Lat   : num  43 43 43 43 43 43 43 43 43 43 ...
 $ Lon   : num  -68.5 -68.5 -68.5 -68.5 -68.5 -68.5 -68.5
                -68.5 -68.5 -68.5 ...
 $ Day   : int  15 46 74 105 135 166 196 227 258 288 ...
 $ h     : num   75.1  90.0  40.7  30.9 107.1 ...
 $ sigma : num  27.0 27.5 25.0 23.9 17.5 ...
 $ z_m   : num  16.0 21.3 10.2 10.7  6.8 ...
 $ B_0   : num  0.335 0.319 0.198 0.248 0.161 ...
 $ P_mB  : num  2.18 2.18 2.58 2.58 2.58 3.56 3.56 3.56 2.52 2.52 ...
 $ alphaB: num  0.123 0.123 0.094 0.094 0.094 0.101 0.101 0.101
                0.059 0.059 ...
 $ P_ZT  : num   396  543  472  543 1275 ...
 $ zp    : num  37.3 36.1 46.9 49.7 34.9 31.9 36.7 37.5 58.8 33.6 ...
 $ Cloud : num  73.2 70.8 65.3 65.5 61.8 58.6 63.1 58.9 48.6 66.9 ...
 $ Yelsub: num  0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 ...
 $ status: int  0 0 0 0 0 0 0 0 0 0 ...
```

It is important to note that the current `dwcpn` program has only simple checks for bad or missing input data. Bad or missing values in the input data frame should be converted to `NA` values and then incomplete records removed (*e.g.*, using `na.omit`) prior to running `dwcpn`. The file `2004maya\_dwcpn.dat` contains a large data set. As an exercise, figure out how to read the data into R, recode missing values, and reset the variable names. You should end up with the data frame shown below.

```
> str(dwcpn.2004maya.in)
'data.frame':   242231 obs. of  13 variables:
 $ depth : num  400 398 396 394 392 389 387 383 379 377 ...
 $ Lat   : num  52 52 52 52 52 52 52 52 52 52 ...
 $ Lon   : num  -53.5 -53.5 -53.4 -53.4 -53.4 ...
 $ Day   : int  128 128 128 128 128 128 128 128 128 128 ...
 $ alphaB: num  0.062 NA 0.099 0.077 0.072 0.054 0.05 0.125
                0.058 0.059 ...
 $ P_mB  : num  2.42   NA 2.16 2.34 1.82 ...
 $ z_m   : num  16.5   NA 22.8 16.8 16.9 ...
 $ B_0   : num  0.715 0.602    NA 1.308 0.837 ...
 $ h     : num  197 264  NA 506 136 ...
 $ sigma : num  16.9   NA 24.9 15.2 13.7 ...
 $ Cloud : num  4.45 4.45 4.45 4.45 4.54 ...
 $ Yelsub: num  0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 ...
 $ SST   : num  0.125 NA 0.125 NA NA NA NA 0.125 0.25 0.125 ...
```

For these data, `na.omit` removes nearly 8000 incomplete records. Note that the number of omitted records are recorded in an attribute:

```
> dwcpn.2004maya.cin=na.omit(dwcpn.2004maya.in)
> str(dwcpn.2004maya.cin)
'data.frame':   234290 obs. of  13 variables:
 $ depth : num  400 383 379 377 373 371 368 367 365 364 ...
 $ Lat   : num  52 52 52 52 52 52 52 52 52 52 ...
 $ Lon   : num  -53.5 -53.3 -53.3 -53.3 -53.3 ...
 $ Day   : int  128 128 128 128 128 128 128 128 128 128 ...
 $ alphaB: num  0.062 0.125 0.058 0.059 0.059 0.118 0.107
                0.113 0.044 0.047 ...
 $ P_mB  : num  2.42 3.03 1.67 1.67 1.67 ...
 $ z_m   : num  16.5 27.3 26.9 13.3 13.3 ...
 $ B_0   : num  0.715 0.214 1.304 0.852 1.978 ...
 $ h     : num  197 415 243 201 902 ...
 $ sigma : num  16.9 16.7 14.4 17.8 17.8 ...
 $ Cloud : num  4.45 4.54 4.63 4.63 4.63 ...
 $ Yelsub: num  0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 ...
 $ SST   : num  0.125 0.125 0.25 0.125 0.125 0.25 0.375
                0.375 0.5 0.375 ...
 - attr(*, "na.action")=Class 'omit'  Named int [1:7941] 2 3
                        4 5 6 7 365 366 367 368 ...
  .. ..- attr(*, "names")= chr [1:7941] "2" "3" "4" "5" ...
```

You can now run `dwcpn` on this file, but it will take many hours to run.

7 Batch processing

The previous sections have provided an introduction to the basic tools, `nlsr::nlxb` and `yaimpute::yai`, used to estimate parameter values from *in situ* data and then use the results for operational ocean production calculations.

8 Installation

Installers for the current versions of R are available for Windows, macOS, and for major linux distributions (including SUSE, Red Hat, and Debian), either from CRAN or from the linux distribution's package sites. After installing the basic R system, it will be necessary to install at least one optional package, `nlsr`. This process is well documented, but linux users will have to ensure that they have also installed any libraries required by add-on packages.

The `nlsr` library will need to installed from source on linux. The `digest` package is required by nslr and will also be installed if not already present.

9 Bugs

Cutting and pasting code from a PDF file viewer into R may give errors when certain characters have been represented in the PDF document using Unicode. Common problems are incorrect string “quote” marks and the “minus” sign. Edit the pasted text by retyping these marks.

The code does only minimal checking of the input data. It is important that the names of the columns in data frames exactly match in spelling and case of the names given here.

When the fitting programs are run on linux you may see warning messages from the X window system. These can generally be ignored.

When running under X, the data editor may fail to refresh the screen after a popup window closes. You can usually refresh the display by minimizing the window and then opening it again.

10 Changes

- 2012-05-01 converted from ConTeXT plus ProTeX to Org mode
- 2012-05-02 update for current nls objects
- 2017-09-12 replace `stats::nls` with `nlsr::nlxb`

11 References

Becker, R.A., J.M. Chambers, and A.R. Wilks (1988). *The new S language: a programming environment for data analysis and graphics*. Wadsworth & Brooks/Cole computer science series. Wadsworth & Brooks/Cole Advanced Books & Software. ISBN: 9780534091934. URL: `https://books.google.ca/books?id=jM1UAAAAYAAJ`.

Ihaka, Ross and Robert Gentleman (1996). “R: A Language for Data Analysis and Graphics”. In: *Journal of Computational and Graphical Statistics* 5.3, pp. 299–314. URL: `http://www.amstat.org/publications/jcgs/`.

Platt, T., C.L. Gallegos, and W.G. Harrison (1980). “Photoinhibition of photosynthesis in natural assemblages of marine phytoplankton”. In: *Journal of Marine Research (USA)* 38, pp. 687–700.

Platt, Trevor et al. (2008). “Operational estimation of primary production at large geographical scales”. In: *Remote Sensing of Environment* 112.8. Earth Observations for Marine and Coastal Biodiversity and Ecosystems Special Issue, pp. 3437–3448. ISSN: 0034-4257. DOI: http://dx.doi.org/10.1016/j.rse.2007.11.018. URL: http://www.sciencedirect.com/science/article/pii/S0034425708001284.

12 Appendix: Functions

```
SGauss <- function(z, B0, h, sigma, zm)
{
   (B0 + h*exp(-(z-zm)^2/(2*sigma*sigma))
/(sigma*sqrt(2*pi)))
}
Production <- function(Irradiance, alpha, PmB)
{
   return ( PmB*(1-exp(-alpha*Irradiance/PmB)))
}
SGplot<-function(data, B0=NA, h=NA, sigma=NA, zm=NA){
plot.sg<-function(z, B, B0, h, sigma, zm ){
  plot(z~B,ylim=c(max(z),0))
  lines(seq(0,max(z))~SGauss(seq(0,max(z)), B0, h, sigma, zm))
  }
with(data,plot.sg(z,B, B0, h, sigma, zm))
}
PIplot<-function(data, alpha, PmB){
plot.pi<-function(P,I, alpha, PmB){
  main<-sprintf("Production: alpha=%f, PmB=%f", alpha, PmB)
  plot(P~I, main=main)
  lines(Production(seq(0,max(I)), alpha, PmB)~seq(0,max(I)))
}
with(data,plot.pi(P,I, alpha, PmB))
}
dwcpn<-function(data, basename, ...){
local({
  write_dwcpn<-function(data, fnm){
     con <- file(fnm,"w")  # open output file connection
     header<-paste('depth Lat Lon Day alphaB P_mB z_m B_0',
   'h sigma Cloud Yelsub',sep=' ')
```

```
     # Fortran-style format
     fmt<-paste('(f7.1,x,f6.2,x,f7.2,x,i3,x,f6.4,x,f6.2,x',
  'f7.2,x,f8.4,x,f6.2,x,f7.2,x,f6.2,x,f4.2)', sep=',')
     cat(file=con, header, fmt, sep='\n')
     # the same format translated to sprintf form
     fmt<-paste('%7.1f %6.2f %7.2f %3i %6.4f %6.2f %7.2f',
'%8.4f %6.2f %7.2f %6.2f %4.2f', sep=' ')
     cat(file=con, sprintf(fmt,data$depth,data$Lat,
 data$Lon,data$Day,data$alphaB,data$P_mB,
 data$z_m,data$B_0,data$h,data$sigma,
 data$Cloud,data$Yelsub),sep='\n')
     close(con)
  }

  run_dwcpn<-function(fnm, ophome='/SGSP/OP'){
     cmd<-paste("dwcpn", " ", fnm, sep='')
     Sys.setenv(OPHOME=ophome)
     system(cmd)
  }

  read_dwcpn<-function(fnm)  {
     # reformat file
     names<-c("Lat","Lon","Day","h","sigma","z_m","B_0",
      "P_mB","alphaB", "P_ZT", "zp", "Cloud",
      "Yelsub", "status")
     return(read.table(fnm,col.names=names,skip=2))
  }

    fnm<-sprintf('%s.dat', basename)
    write_dwcpn(data,fnm)
    run_dwcpn(fnm)
    fnm<-sprintf('%s.out', basename)
    read_dwcpn(fnm)
})
}
x <- c(
536.8, 2.80,
408.5, 2.51,
356.3, 2.31,
313.5, 2.55,
161.5, 2.43,
133.0, 2.14,
111.2, 2.10,
98.8, 1.95,
89.3, 1.71,
79.3, 1.47,
53.2, 1.36,
42.8, 1.06,
38.0, 0.98,
34.2, 0.88,
30.9, 0.68,
27.6, 0.58,
19.0, 0.46,
15.7, 0.38,
13.8, 0.32,
12.2, 0.21,
11.1, 0.16,
9.9, 0.14,
6.6, 0.10,
5.6, 0.07,
4.9, 0.05,
2.2, 0.02,
1.9, 0.01)
dim(x) <- c(2, 27)
PI.df <- as.data.frame(t(x))
names(PI.df) <-  c("I", "P")
profile.df <- data.frame(z=seq(0, 100, by=10),
     B=c(0.6, 0.7, 1.0, 1.6, 1.3, 0.6, 0.48, 0.52, 0.50, 0.4, 0.6))
```