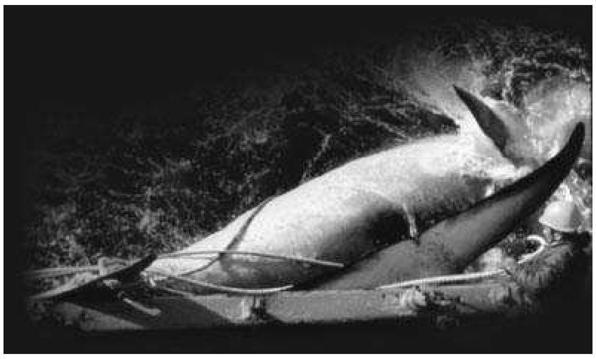
Norsk Regnesentral
NORWEGIAN COMPUTING CENTER

Note

Updated documentation of a Fortran 77 subroutine implementing the catch limit algorithm, January 2006



Note no SAMBA/06/06

Authors Ragnar Bang Huseby
Magne Aldrin

Date January 2006

Norwegian Computing Center

Norsk Regnesentral
Norwegian Computing Center
Postboks 114, Blindern
NO-0314 Oslo, Norway

Besøksadresse
Office address
Gaustadalléen 23
NO-0373 Oslo, Norway

Telefon · telephone
(+47) 22 85 25 00
Telefaks · telefax
(+47) 22 69 76 60

Internett · internet
www.nr.no
E-post · e-mail
nr@nr.no

Title	**Updated documentation of a Fortran 77 subroutine implementing the catch limit algorithm, January 2006**
Authors	**Ragnar Bang Huseby** <ragnar.huseby@nr.no> **Magne Aldrin** <magne.aldrin@nr.no>
Date	January 2006
Publication number	SAMBA/06/06

Abstract

Keywords	
Target group	
Availability	
Project	
Project number	
Research field	
Number of pages	27
© Copyright	Norwegian Computing Center

Contents

1 Introduction

The Scientific Committee of The International Whaling Commission has tested various procedures on simulated population and catch histories. In 1991 the Committee chose one procedure, proposed by Cooke, as the core element of the so called “Revised Management Procedure”.

This procedure, as specified in (Rep. Int. Whal. Commn. 44, Annex H.), has been implemented by the Norwegian Computing Center. The program, called `rmp`, was described in (Fenstad et al., 1993). From this program, the module implementing the catch limit algorithm has been extracted. The module was modified in June 1999, and further changes of the code have been made in June 2000 and November 2000. The version of November 2000 will be described in this note.

The catch limit algorithm is reviewed in Section 2. In Section 3, we describe how the catch limit is computed in our implementation, and in Section 4, we review the numerical analysis methods used.

Appendix A contains a manual description of the subroutine computing the catch limit. Appendix B contains a list of the subroutines of the module. The difference between the various versions of the module is described in Appendix C.

2 Catch limit algorithm

In this section, the catch limit algorithm is reviewed. We use the same notation as in (Rep. Int. Whal. Commn. 44, Annex H.). The input data consists of the time series of historic annual catches and the time series of absolute abundance estimates along with the information matrix of the logarithm of the estimates. In our implementation we assume that

1. the abundance estimates are positive, and
2. the information matrix of the logarithm of the estimates is nonnegative definite.

The internal population model of the catch limit algorithm is defined by the following dynamics

$$\begin{aligned} P_0 &= \frac{P_T}{D_T}, \\ P_{t+1} &= P_t - C_t + 1.4184\,\mu\, P_t(1-(\frac{P_t}{P_0})^2) \qquad (0 \le t < T), \end{aligned} \tag{1}$$

where

⋆ P_t is the population size in numbers at the beginning of year t

⋆ C_t is the catch in numbers in year t
⋆ $D_T = P_T/P_0$ is the ratio of the population size at the beginning of year T to the population size at the beginning of year zero, denoted stock depletion
⋆ Year zero is the first year of the historic catch series used in assessments
⋆ Year T is the year the catch limit is to be applied (i.e. the first year of an assessment cycle). This is assumed to be the year immediately following the last year of historic catch series used in the assessments
⋆ μ is a parameter describing the productivity.

In this model, μ and D_T are regarded as fixed, but unknown parameters, which together determine the population history, as long as there has been any catches. (In the case of no previous catches, a nominal catch of one whale in year 0 is assumed.)

The abundance estimates are assumed to be log-normally distributed with a given information matrix for the log estimates, estimated from the survey data. The formula for the data likelihood is

$$\text{Likelihood}(\mu, D_T, b) \propto \exp\Big(-1/2(\mathbf{a} - \mathbf{p} - \beta\mathbf{1})' H(\mathbf{a} - \mathbf{p} - \beta\mathbf{1})\Big) \tag{2}$$

where
⋆ $\mathbf{a}$ is the vector of logarithms of the estimates of population size by year;
⋆ $\mathbf{p}$ is the vector of logarithms of the modeled annual population sizes for the years with population estimates, $p_t = \ln(P_t)$;
⋆ β is the logarithm of the bias parameter, thus $b = \exp(\beta)$;
⋆ H is the information matrix of the $\mathbf{a}$ vector. If H is nonsingular, $H = V^{-1}$ where V is (an estimate of) the covariance matrix of the vector $\mathbf{a}$.

The parameters μ, D_T, and b are assigned a prior distribution which is uniform over the region

$$[\mu_{\min}, \mu_{\max}] \times [D_{T,\min}, D_{T,\max}] \times [b_{\min}, b_{\max}], \tag{3}$$

where $\mu_{\min}$, $\mu_{\max}$, $D_{T,\min}$, $D_{T,\max}$, $b_{\min}$, and $b_{\max}$ are constants. Typical values are $\mu_{\min} = 0.0$, $\mu_{\max} = 0.05$, $D_{T,\min} = 0.0$, $D_{T,\max} = 1.0$, $b_{\min} = 0.0$, and $b_{\max} = 1.6667$. The joint likelihood function of the parameters μ, D_T, and b is now determined. It is given as follows:

$$\text{Posterior}(\mu, D_T, b) \propto \text{Prior}(\mu, D_T, b) \cdot \text{Likelihood}(\mu, D_T, b)^s, \quad s = 1/16 \tag{4}$$

The presence of a deflation parameter $0 < s < 1$ down-weights the survey information relative to a strict Bayesian approach.

The internal catch limit is the following function of γ, μ, D_T, and P_T:

$$L_T = \begin{cases} 0 & \text{if } D_T \leq IPL \\ \gamma\mu(D_T - IPL)P_T & \text{if } D_T > IPL \end{cases} \tag{5}$$

where the slope γ and the internal protection level IPL are control parameters. Typical values of γ and IPL have been 3 and 0.54, respectively. The internal catch limit can be regarded as the catch limit in the hypothetical case of perfect knowledge of population parameters and size. However, in the Bayesian formalism, it is regarded as a random variable, with marginal posterior distribution obtained from the joint posterior distribution of (μ, D_T, b). The actual catch limit z is defined as a certain percentile of the marginal distribution of L_T. Hence z satisfies

$$P(L_T < z|data) \leq \alpha \leq P(L_T \leq z|data) \tag{6}$$

for a given α. A typical value of α is 0.4102.

3 Computation details

Change of variables: Computation of the catch limit involves integration of the right-hand side of (4) over various subsets of the parameter space. In order to avoid solving for the population history for each functional evaluation, the calculation is based on a change of variables from (μ, D_T, b) to (μ, p_0, b) where $p_0 = \ln(P_0)$. The Jacobi determinant, $J(\mu, p_0, b)$, of the mapping from (μ, p_0, b) to (μ, D_T, b) is defined by

$$J(\mu, p_0, b) = \begin{vmatrix} \frac{\partial\mu}{\partial\mu} & \frac{\partial\mu}{\partial p_0} & \frac{\partial\mu}{\partial b} \\ \frac{\partial D_T}{\partial\mu} & \frac{\partial D_T}{\partial p_0} & \frac{\partial D_T}{\partial b} \\ \frac{\partial b}{\partial\mu} & \frac{\partial b}{\partial p_0} & \frac{\partial b}{\partial b} \end{vmatrix} \tag{7}$$

where $|A|$ means the determinant of the matrix A. It follows that

$$J(\mu, p_0, b) = \frac{\partial P_T}{\partial P_0} - D_T. \tag{8}$$

In order to compute $J(\mu, p_0, b)$ we use the recursion

$$\begin{aligned} \frac{\partial P_0}{\partial P_0} &= 1, \\ \frac{\partial P_{t+1}}{\partial P_0} &= (1 + R - 3R(\frac{P_t}{P_0})^2)\frac{\partial P_t}{\partial P_0} + 2R(\frac{P_t}{P_0})^3 \quad (0 \leq t < T), \end{aligned} \tag{9}$$

where $R = 1.4184\,\mu$.

It is implicitly assumed that p_0 is a monotone function of μ when D_T is fixed. This will be the case if $J(\mu, p_0, b) > 0$ everywhere except possibly on the boundary, or in the limit as $P_0 \rightarrow \infty$. This has not been proved in the strict sense. It has, however, always turned out to be the case in our numerical computations. Thus, there is sufficiently strong numerical evidence to regard the question as settled for all practical purposes.

Splitting the integral over the parameter space: We need to find the integral of the right-hand side of (4) over the region defined by (3). This integral is given by

$$\int_{\mu_{\min}}^{\mu_{\max}} \int_{D_{T,\min}}^{D_{T,\max}} \int_{b_{\min}}^{b_{\max}} \text{Prior}(\mu, D_T, b) \cdot \text{Likelihood}(\mu, D_T, b)^s db\, dD_T\, d\mu. \tag{10}$$

This integral is also equal to

$$\int_{\mu_{\min}}^{\mu_{\max}} \int_{-\infty}^{\infty} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b) db\, dp_0\, d\mu, \tag{11}$$

where

$$f(\mu, p_0, b) = \text{Prior}(\mu, D_T, b) \cdot \text{Likelihood}(\mu, D_T, b)^s \cdot |J(\mu, p_0, b)|, \tag{12}$$

and $|J(\mu, p_0, b)|$ is the absolute value of the Jacobi determinant. Note that D_T is a function of (μ, p_0). In the computation, it is convenient to split the integral at $D_T = IPL$. Thus, the integral is equal to $I_{lower} + I_{upper}$, where

$$I_{lower} = \int_{\mu_{\min}}^{\mu_{\max}} \int_{-\infty}^{p_{0,split}(\mu)} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b) db\, dp_0\, d\mu, \tag{13}$$

and

$$I_{upper} = \int_{\mu_{\min}}^{\mu_{\max}} \int_{p_{0,split}(\mu)}^{\infty} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b) db\, dp_0\, d\mu, \tag{14}$$

where $p_{0,split}(\mu)$ is the value of p_0 such that

$$D_T = IPL \tag{15}$$

for a given μ, and IPL is as in (5).
I_{lower} is split further by splitting the range $(-\infty, p_{0,split}(\mu)]$ into the two intervals $(-\infty, p_{0,lowmid}(\mu)]$ and $[p_{0,lowmid}(\mu), p_{0,split}(\mu)]$, where $p_{0,lowmid}(\mu)$ is the value of p_0 such that

$$D_T = \frac{4}{5} D_{T,\min} + \frac{1}{5} IPL \tag{16}$$

for a given μ. By a change of variable from p_0 to u where

$$p_0 = p_{0,lowmid}(\mu) + (p_{0,lowmid}(\mu) - p_{0,split}(\mu))(\frac{2}{1-u} - 1), \tag{17}$$

and

$$\frac{dp_0}{du} = (p_{0,lowmid}(\mu) - p_{0,split}(\mu))\frac{2}{(1-u)^2}, \tag{18}$$

the integral over $(-\infty, p_{0,lowmid}(\mu)]$ is transformed to an integral over the finite interval $[-1, 1]$. Thus $I_{lower} = I^-_{lower} + I^+_{lower}$, where

$$I^-_{lower} = \int_{\mu_{\min}}^{\mu_{\max}} \int_{-1}^{1} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b)\frac{dp_0}{du} db\, du\, d\mu, \tag{19}$$

and

$$I^+_{lower} = \int_{\mu_{\min}}^{\mu_{\max}} \int_{p_{0,lowmid}(\mu)}^{p_{0,split}(\mu)} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b) db\, dp_0\, d\mu. \tag{20}$$

Similarly, I_{upper} can be written $I_{upper} = I^-_{upper} + I^+_{upper}$, where

$$I^-_{upper} = \int_{\mu_{\min}}^{\mu_{\max}} \int_{p_{0,split}(\mu)}^{p_{0,highmid}(\mu)} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b) db\, dp_0\, d\mu, \tag{21}$$

and

$$I^+_{upper} = \int_{\mu_{\min}}^{\mu_{\max}} \int_{-1}^{1} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b)\frac{dp_0}{dv} db\, dv\, d\mu, \tag{22}$$

where $p_{0,highmid}(\mu)$ is the value of p_0 such that

$$D_T = \frac{4}{5}D_{T,\max} + \frac{1}{5}IPL \tag{23}$$

for a given μ,

$$p_0 = p_{0,highmid}(\mu) + (p_{0,highmid}(\mu) - p_{0,split}(\mu))(\frac{2}{1-v} - 1), \tag{24}$$

and

$$\frac{dp_0}{dv} = (p_{0,highmid}(\mu) - p_{0,split}(\mu))\frac{2}{(1-v)^2}. \tag{25}$$

Setting up an equation for the catch limit: In order to find z such that (6) is satisfied, we need to compute $P(L_T \leq z|data)$ for various values of z. $P(L_T \leq z|data)$ is equal to

$$\frac{\int_{\mu_{\min}}^{\mu_{\max}}\int_{D_{T,\min}}^{D_{T,z}(\mu)}\int_{b_{\min}}^{b_{\max}} \text{Prior}(\mu, D_T, b) \cdot \text{Likelihood}(\mu, D_T, b)^s db\, dD_T\, d\mu}{\int_{\mu_{\min}}^{\mu_{\max}}\int_{D_{T,\min}}^{D_{T,\max}}\int_{b_{\min}}^{b_{\max}} \text{Prior}(\mu, D_T, b) \cdot \text{Likelihood}(\mu, D_T, b)^s db\, dD_T\, d\mu}, \tag{26}$$

where $D_{T,z}(\mu)$ is the value of D_T such that

$$L_T = z \tag{27}$$

for a given μ. L_T is the internal catch limit defined by (5). The denominator of (26) is equal to $I_{lower} + I_{upper}$. If $z = 0$, the numerator is equal to I_{lower}. If $z > 0$, the numerator is equal to $I_{lower} + I_z$, where I_z is given by

$$I_z = \int_{\mu_{\min}}^{\mu_{\max}} \int_{p_{0,split}(\mu)}^{p_{0,z}(\mu)} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b) db\, dp_0\, d\mu, \tag{28}$$

$p_{0,split}(\mu)$ is defined by (15), and $p_{0,z}(\mu)$ is the value of p_0 such that (27) is satisfied for a given μ. It follows that $z = 0$ is the solution of (6) if $I_{lower}/(I_{lower} + I_{upper}) \geq \alpha$. Otherwise, z satisfies

$$\frac{I_{lower} + I_z}{I_{lower} + I_{upper}} = \alpha. \tag{29}$$

Approximating the catch limit: We are now ready to describe a procedure that computes an approximation of the catch limit. In this procedure, the integrals I^-_{lower}, I^+_{lower}, I^-_{upper}, I^+_{upper}, and I_z defined by (19), (20), (21), (22), and (28), respectively, are calculated by numerical integration. The integrals are evaluated as iterated integrals, and the order of integration is as indicated in the equations above. Each iterated integral is approximated by an n-point Gauss-Legendre integration rule, (Davis and Rabinowitz, 1975). The integer n, which is kept fixed in this procedure, is the number of functional evaluations in the approximation. Thus, the approximation can be written as a sum

$$\sum_{i=1}^{n} w_i \, g(x_i), \tag{30}$$

where the w_i's are weights, the x_i's are abscissas, and g is the integrand. The weights and the abscissas depend only on the interval of integration and not on the function to be integrated. For a review of the Gauss-Legendre integration rules, see Section 4.1.
The approximation procedure can be divided into the following steps.

1. Calculate the weights and the abscissas in the Gauss-Legendre integration rule approximating the b-integral.

2. Calculate the weights and the abscissas in the Gauss-Legendre integration rule approximating the μ-integral.

3. For each abscissa in the Gauss-Legendre integration rule approximating the μ-integral, find $p_{0,split}(\mu)$ defined by (15). This equation is solved numerically by Brent's method, (Press et al., 1992). For a brief review of Brent's method, see Section 4.2. In order to find the solution, D_T is evaluated for various values of p_0 by using (1). It is assumed that $-5 \leq p_{0,split}(\mu) \leq 50$.

4. For each abscissa in the Gauss-Legendre integration rule approximating the μ-integral, find $p_{0,lowmid}(\mu)$ defined by (16). This equation is solved in the same way as in Step 3. It is assumed that $-5 \leq p_{0,lowmid}(\mu) \leq 50$.

5. For each abscissa in the Gauss-Legendre integration rule approximating the μ-integral: calculate the weights and the abscissas in the Gauss-Legendre integration rules approximating the v-integral in (20) and the u-integral in

(19). Each weight in the u-integral is multiplied by $\frac{dp_0}{du}$ evaluated at the corresponding abscissa. $\frac{dp_0}{du}$ is given by (18).

6. Evaluate an approximation of $I_{lower} = I^-_{lower} + I^+_{lower}$. Each integral on the right-hand side is approximated by a triple sum. In order to find the sums the function f defined by (12) is evaluated at various points. At the points satisfying $-5 \leq p_0 \leq 50$, the population history, see (1), the Jacobi determinant, see (8), and the right-hand side of (2), are calculated. Concerning the calculation of the population history, there are some exceptions that occur if P_0 is large or the population size becomes small, see the documentation of the subroutine `pforw` in Appendix B. The right-hand side of (2) can be written as

$$\exp(-\frac{1}{2}(D_3 - \beta\, D_2 + \beta^2 D_1)) \tag{31}$$

where

$$D_1 = \sum_{i=1}^{n}\sum_{j=1}^{n} H_{i,j}, \tag{32}$$

$$D_2 = \sum_{i=1}^{n}\sum_{j=1}^{n} H_{i,j}\,(a_{y_i} - p_{y_i}), \tag{33}$$

$$D_3 = \sum_{i=1}^{n}\sum_{j=1}^{n} H_{i,j}\,(a_{y_i} - p_{y_i})\,(a_{y_j} - p_{y_j}), \tag{34}$$

a_{y_i} and p_{y_i} are the logarithms of the abundance estimate and the modeled population size, respectively, by year y_i, and $H_{i,j}; i = 1, 2, \ldots, n; j = 1, 2, \ldots, n$ are the entries of the information matrix H. The sums D_1, D_2, and D_3 are computed only once for each (μ, p_0). At the points where either $p_0 < -5$ or $p_0 > 50$, $f(\mu, p_0, b)$ is set to zero.

7. For each abscissa in the Gauss-Legendre integration rule approximating the μ-integral, find $p_{0,highmid}(\mu)$ defined by (23). This equation is solved in the same way as in Step 3. It is assumed that $-5 \leq p_{0,highmid}(\mu) \leq 50$.

8. Calculate the weights and the abscissas relevant for the computation of I^-_{upper} and I^+_{upper}. This is done in the similar way as in Step 5. Gauss-Legendre integration rules approximates the p_0-integral in (21) and the v-integral in (22).

9. Evaluate an approximation of $I_{upper} = I^-_{upper} + I^+_{upper}$. This is similar to Step 6.

10. If $I_{lower}/(I_{lower} + I_{upper}) \geq \alpha$, the catch limit approximation is zero. Otherwise, the catch limit approximation is the solution of (29) found by Brent's method.

    It is assumed that the solution is in $[0, \mu_{\max} A_*]$, where A_* is either A_τ, the most recent abundance estimate, or $A_{\tau-1}$, the second most recent abundance estimate. If $A_\tau < A_{\tau-1}$ and the variance of the second most recent abundance estimate is smaller than the variance of the most recent abundance estimate, $A_* = A_{\tau-1}$. Otherwise, $A_* = A_\tau$.

In order to compute the left-hand side of (29), the following tasks must be completed.

a. For each abscissa in the Gauss-Legendre integration rule approximating the μ-integral, find $p_{0,z}(\mu)$ defined by (27). This equation is solved in the same way as in Step 3.

b. Calculate the weights and the abscissas in the Gauss-Legendre integration rules approximating the p_0-integral in (28).

c. Evaluate an approximation of I_z. This is similar to Step 6.

In extreme cases when $f(\mu, p_0, b) \approx 0$ except on a very small subset of the region of integration, the computed approximation of $I_{lower} + I_{upper}$ might be zero. In that case, the procedure fails to compute an approximation of the catch limit. This type of failure becomes less likely as n grows.

Computing the catch limit by an iterative algorithm: The procedure above approximating the catch limit using n-point Gauss-Legendre integration rules is carried out for $n = 8, 16, 32, 64, 128, 256, 512, 1024$, or until the difference between two successive approximations becomes less than a tolerance specifying the required accuracy.

Error handling: Our implementation does not handle the most extreme cases. If there is evidence that the catch limit cannot be computed to the required accuracy, this will be reported through the output of the main routine of the module. For further details, see the specifications of the parameter `IFAIL` in Appendix A.

4 Description of numerical analysis methods

4.1 Gauss-Legendre integration rules

In this section, we give a brief review of the Gauss-Legendre integration rules. For more details, see e.g. Section 2.7 in (Davis and Rabinowitz, 1975).

A Gauss-Legendre integration rule is a way of approximating the integral of a function over an interval. The approximation is of the form

$$\int_a^b g(x)dx \approx \sum_{i=1}^{n} w_i\, g(x_i). \tag{35}$$

The weights w_i's and the abscissas x_i's are chosen such that

$$\int_a^b q(x)dx = \sum_{i=1}^{n} w_i\, q(x_i). \tag{36}$$

whenever q is a polynomial of degree $\leq 2n-1$. This is the basic idea of Gauss-Legendre integration rules.
The x_i's are the zeros of the polynomial p_n^*, where the polynomials $p_0^*, p_1^*, \ldots$ satisfy the following conditions.

1. p_n^* is a polynomial of degree n.
2. $\int_a^b (p_n^*(x))^2 dx = 1$.
3. $\int_a^b p_m^*(x) p_n^*(x) dx = 0$ whenever $m \neq n$.

The w_i's are given by

$$w_i = -\frac{k_{n+1}}{k_n} \frac{1}{p_{n+1}^*(x_i) p_n^{*\prime}(x_i)} \tag{37}$$

where k_n is the coefficient of x^n in $p_n^*(x)$.
The subroutine GRULE at page 369 in (Davis and Rabinowitz, 1975) is used in our implementation. This subroutine computes the $m = [(n+1)/2]$ nonnegative abscissas x_i's and the corresponding weights w_i's of the n-point Gauss-Legendre integration rule when the interval of integration is $[-1, 1]$.

In order to find the abscissas x_i''s and the weights w_i''s in the general case when the interval of integration is $[a, b]$, we use the fact that the abscissas are located symmetrically in the interval $[a, b]$ and the weights corresponding to symmetric points are equal. Then the following relations are valid for $i = 1, \ldots, m$:

$$\begin{aligned} x_i' &= c - d\, x_i, \\ x_{m+i}' &= c + d\, x_{m-i+1}, \\ w_i' &= d\, w_i, \\ w_{m+i}' &= d\, w_{m-i+1}, \end{aligned} \tag{38}$$

where $c = (a+b)/2$ and $d = (b-a)/2$.

4.2 Brent's method for solving equations

In this section, we consider the problem of finding the value of x such that $g(x) = c$ where g is a function of one variable. This problem is equivalent to the problem of finding x such that $f(x) = 0$, where $f(x) = g(x) - c$. Brent's method solves the latter problem numerically. The method combines root bracketing, bisection, and inverse quadratic interpolation, (Press et al., 1992). In our implementation we use the function zbrent in (Press et al., 1992) with a slight modification. In order to reduce the amount of computation we first search for a solution in a narrow interval. If we do not succeed, we search for a solution in a broader interval. In the implementation of (Press et al., 1992) there is no possibility of extending the search interval.

References

Davis, P. and Rabinowitz, P. (1975). *Methods of numerical integration*. Academic Press.

Fenstad, A., Helgeland, J., Aldrin, M., and Volden, R. (1993). High accuracy computer program for the IWC catch limit algorithm. In *SC/45/Mg3, IWC SC Annual Meeting*, Kyoto, Japan.

Martin, Reinsch, and Wilkinson (1968). Num. Math. 11.

Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (1992). *Numerical Recipes in FORTRAN*. Cambridge University Press, 2nd edition.

Reinsch (1973). Comm. ACM 16.

Rep. Int. Whal. Commn. 44, Annex H. (1994).

Smith, B., Boyle, J., Dongarra, J., Garbow, B., Ikebe, Y., Klema, V., and Moler, C. (1976). Matrix Eigensystem Routines. In *EISPACK Guide Lecture Notes in Computer Science*, volume 6. Springer-Verlag.

A CATCHLIMIT - Manual description of the subroutine

Purpose: Calculate the catch limit for a single area according to the algorithm of Section 2.

Restrictions: We assume that the abundance estimates are positive, and the information matrix of the logarithm of the estimates is nonnegative definite. The subroutine is unable to compute the catch limit in extreme cases.

Files: The file `xrmpSub.f` contains the module implementing the catch limit algorithm including the subroutine `CATCHLIMIT`. The file `xrmpSub_inc.f` contains definitions of some common blocks used by the module and must be included.

Specification:

```
 SUBROUTINE CATCHLIMIT(NUM,ABDIM,CATCH,ABEST,INFOMATRX,R8WORKSPACE,
*     AB_YEARS,IN_PPROB,IN_MU_MIN,IN_MU_MAX,IN_DT_MIN,IN_DT_MAX,
*     IN_B_MIN,IN_B_MAX,IN_PLEVEL,IN_PSLOPE,IN_PSCALE,IN_NOF_RULE,
*     OUT_QUOTA,accQuota,outDiff,npRule,
*     POP,DEVPOP,IN_INFOLEVEL,IN_IOUT,IFAIL)
```

Parameters:

1. `NUM - integer` `Input`
   On entry: The length of the catch history. Actual length of `CATCH` array

2. `ABDIM - integer` `Input`
   On entry: Number of years with nonzero abundance estimates for the area in question.

3. `CATCH(NUM) - real*8 array` `Input`
   On entry: `CATCH(Y)` is the historic catch in year `Y`; `Y = 1,2,...,NUM`.
   Corresponds to C_t in (1).
   If there has been any catch, `Y = 1` corresponds to the first year of catch ($t = 0$ in (1)).
   `Y = NUM` corresponds to last premanagement year ($t = T - 1$ in (1)).
   If there has been no catch, `NUM` should be equal to 1.
   If `NUM=1` and `CATCH(1)=0`, `CATCH(1)` is set to 1.
   Constraint: `CATCH(1) > 0 if NUM > 1`.

4. `ABEST(ABDIM) - real*8 array` Input
   On entry: `ABEST(I)` is the absolute abundance estimates in year `AB_YEARS(I); I = 1,2,...,A`
   The vector of the logarithms of the entries in this array corresponds to a in (2).
   Constraint: `ABEST(I) > 0`.

5. `INFOMATRX(ABDIM*(ABDIM+1)/2) - real*8 array` Input
   On entry: `INFOMATRX(I)` contains the lower triangle of the matrix H in (2) stored row-wise.
   Constraint: H is symmetric and nonnegative definite.

6. `R8WORKSPACE(ABDIM*(ABDIM+7)/2) - real*8 array` Workspace
   Workspace needed to determine whether the matrix stored in `INFOMATRX` is nonnegative definite.

7. `AB_YEARS(ABDIM) - integer array` Input
   On entry: `AB_YEARS(I)` is the year of the absolute abundance estimate `ABEST(I)`.
   If `AB_YEARS(I) < 1`, the corresponding abundance estimate is treated as if the sighting was performed in year 1.
   Constrains: `AB_YEARS(I) < NUM+1`,
   `AB_YEARS(1) < AB_YEARS(2) < ... < AB_YEARS(ABDIM)`.

8. `IN_PPROB - real*8` Input
   On entry: Probability level for distribution of L_T.
   Corresponds to α in (6).
   Typical value: 0.4102

9. `IN_MU_MIN - real*8` Input
   On entry: Minimum value of productivity parameter.
   Corresponds to $\mu_{\min}$ in (3).
   Typical value: 0.0
   Constraint: `IN_MU_MIN` is nonnegative.

10. `IN_MU_MAX - real*8` Input
    On entry: Maximum value of productivity parameter.
    Corresponds to $\mu_{\max}$ in (3).
    Typical value: 0.05
    Constraints: `IN_MU_MAX` is not less than 10^{-20} and
    `IN_MU_MAX` is not less than `IN_MU_MIN`.

11. `IN_DT_MIN - real*8` Input
    On entry: Minimum value for stock depletion.
    Corresponds to $D_{T,\min}$ in (3).

Typical value: 0.0
Constraint: `IN_DT_MIN` is nonnegative.

12. `IN_DT_MAX - real*8` `Input`
On entry: Maximum value for stock depletion.
Corresponds to $D_{T,\max}$ in (3).
Typical value: 1.0
Constraint: `IN_DT_MAX` is not less than `IN_DT_MIN`.

13. `IN_B_MIN - real*8` `Input`
On entry: Minimum bias.
Corresponds to $b_{\min}$ in (3).
Typical value: 0.0
Constraint: `IN_B_MIN` is nonnegative.

14. `IN_B_MAX - real*8` `Input`
On entry: Maximum bias.
Corresponds to $b_{\max}$ in (3).
Typical value: 1.6667
Constraints: `IN_B_MAX` is not less than 10^{-20} and
`IN_B_MAX` is not less than `IN_B_MIN`.

15. `IN_PLEVEL - real*8` `Input`
On entry: Internal protection level.
Corresponds to IPL in (5).
Typical value: 0.54
Constraint: $D_{T,\min} \geq IPL \geq D_{T,\max}$.

16. `IN_PSLOPE - real*8` `Input`
On entry: Catch control slope.
Corresponds to γ in (5).
Typical value: 3

17. `IN_PSCALE - real*8` `Input`
On entry: Scaling factor.
The scaling factor is equal to $\frac{1}{\sqrt{s}}$ where s corresponds to the deflation parameter in (4).
Typical value: 4

18. `IN_NOF_RULE - integer` `Input`
On entry: The maximum number of iterations allowed to compute the catch limit. In iteration i, the catch limit is approximated by a 2^{2+i}-point Gauss-Legendre integration rule.
Typical value: 8

19. `OUT_QUOTA - real*8` Output
On exit: Calculated catch limit.

20. `accQuota - real*8` Input
On entry: Tolerance specifying the required accuracy. The iterative algorithm terminates if the difference between two successive approximations of the catch limit (determined by $\frac{n}{2}$-point and n-point Gauss-Legendre integration rules, respectively) is less or equal to `accQuota`.
The approximate solution of (29) is determined such that its accuracy is $0.25 \cdot$ `accQuota`.
Typical value: 0.2.

21. `outDiff - real*8` Output
On exit: Achieved accuracy, that is the difference between the last two approximations of the catch limit.

22. `npRule - integer` Output
On exit: The number of points used in the numerical integration in the last iteration.

23. `POP(0:NUM+1) - real*8 array` Workspace
Various population size trajectories. Corresponds to P_t in (1).

24. `DEVPOP(ABDIM) - real*8 array` Workspace
Difference between abundance estimate and population size at years with abundance estimates for various trajectories.

25. `IN_INFOLEVEL - integer` Input
On entry: Parameter controlling the level of intermediate printout produced by this module. The larger value, the more printout.
Typical values: 0 - no printout,
1 - possible warnings,
2 - as 1 + print each catch limit approximation,
3 - as 2 + print value of integrals,
4 - as 3 + print some integration limits,
5 - as 4 + print input arrays,
6 - as 5 + print D_1, D_2, and D_3 in (32-34),
7 - as 6 + print likelihood and density values.

26. `IN_IOUT - integer` Input
On entry: Unit determining file for intermediate printout.

27. `IFAIL - integer` Input/Output
On entry: If the user sets `IFAIL` to 0 before calling the routine, execution of

the program will terminate if the routine detects an error. Before the program is stopped, an error message is output. If the user sets IFAIL to -1 or 1 before calling the routine, the control is returned to the calling program if the routine detects an error. If IFAIL = -1, an error message is output before the control is returned.
On exit: If IFAIL = 0, no error is detected.
If IFAIL = 2, NUM < 1.
If IFAIL = 3, ABDIM < 1.
If IFAIL = 4, NUM > 1 and CATCH(1) is not positive.
If IFAIL = 5, ABEST(I) is not positive for some I.
If IFAIL = 6, the information matrix of the logarithm of the abundance estimates is not nonnegative definite (At least one of the eigenvalues is negative). Due to numerical inaccuracy a singular matrix may be declared as not being nonnegative definite. In such cases, however, the magnitude of the lowest eigenvalue computed by the module is small. This eigenvalue is printed if IN_INFOLEVEL is positive.
If IFAIL = 7, AB_YEARS(I) > NUM for some I, or the sequence
AB_YEARS(I); I=1,...,ABDIM; is not strictly increasing.
If IFAIL = 8, IN_PPROB < 0 or IN_PPROB > 1.
If IFAIL = 9, MU_MIN > MU_MAX, MU_MIN < 0, or MU_MAX < 10^{-20}.
If IFAIL = 10, DT_MIN > DT_MAX or DT_MIN < 0.
If IFAIL = 11, B_MIN > B_MAX, B_MIN < 0, or B_MAX < 10^{-20}.
If IFAIL = 12, IN_PLEVEL < DT_MIN or IN_PLEVEL > DT_MAX.
If IFAIL = 13, accQuota is not positive.
If IFAIL = 14, nmax in include file is less than the number of rule points.
If IFAIL = 15, possible inaccuracies in computed population size history.
If IFAIL = 16, P_T becomes larger than $0.5 \cdot 10^{30}$.
If IFAIL = 17, the Jacobi determinant $J(\mu, p_0, b)$ becomes negative at some point.
If IFAIL = 18, for some μ it was not possible to find $p_{0,split}(\mu)$ defined by (15).
If IFAIL = 19, for some μ it was not possible to find either $p_{0,lowmid}(\mu)$ defined by (16) or $p_{0,highmid}(\mu)$ defined by (23).
If IFAIL = 20, for some μ it was not possible to find the integration interval of the p_0-integral.
If IFAIL = 21, the value of z in (28) becomes negative.
If IFAIL = 22, the catch limit could not be computed because the computed approximation of (10) is zero.
If IFAIL = 23, it was not possible to solve the equation for the catch limit.
If IFAIL = 24, the required accuracy was not reached.
If IFAIL = -2, the input value of IFAIL is illegal. It is assumed that IFAIL

value should be 0.

B List of subroutines

The module contains the following subroutines and functions.

1. `SUBROUTINE CATCHLIMIT` - Main subroutine and gateway to the module. Performs some tests on input parameters. Calls `checkdat` and `calc_quota`.

2. `SUBROUTINE checkdat` - Checks that the input arrays are legal.

3. `SUBROUTINE checkposdef` - Checks that the information matrix is nonnegative definite.

4. `SUBROUTINE rsp` - calls `tred3` and `tqlrat` to find the eigenvalues of a real symmetric packed matrix. This subroutine comes from the eigensystem package EISPACK, (Smith et al., 1976). The part of the original subroutine that is concerned with eigenvectors is omitted.

5. `SUBROUTINE tred3` - reduces a real symmetric matrix, stored as a one-dimensional array, to a symmetric tridiagonal matrix using orthogonal similarity transformations. This subroutine is a translation of the Algol procedure `tred3` in (Martin et al., 1968). This subroutine comes from the eigensystem package EISPACK, (Smith et al., 1976).

6. `SUBROUTINE tqlrat` - finds the eigenvalues of a symmetric tridiagonal matrix by the rational QL method. This subroutine is a translation of the Algol procedure `tqlrat` in (Reinsch, 1973). This subroutine comes from the eigensystem package EISPACK, (Smith et al., 1976). Calls `epslon` and `pythag`.

7. `REAL*8 FUNCTION epslon` - estimates unit roundoff in quantities of a certain size. This function comes from the eigensystem package EISPACK, (Smith et al., 1976).

8. `REAL*8 FUNCTION pythag` - finds $\sqrt{a^2 + b^2}$ without overflow or destructive underflow. This function comes from the eigensystem package EISPACK, (Smith et al., 1976).

9. `SUBROUTINE calc_quota` - This is the shell of the iterative algorithm for computing the catch limit, see Section 3. Calls `putgauss` (Step 1). Calls `setSplit` (Step 2 and Step 3). Calls `halfInt` in order to compute approximations of I_{lower} and I_{upper} (Steps 4-9). Calls `zbrent` in order to find the zero of the function `fract` (Step 10).

10. `REAL*8 FUNCTION lhood` - Computes the scaled likelihood (the right-hand side of (4)) for a set of parameters.

11. `REAL*8 FUNCTION dens` - Integrates the scaled likelihood (the right-hand side of (4)) with respect to the bias parameter b. Multiplies the result by the Jacobi determinant of the transformation in (8). The result is a function of p_0 and μ. In the exceptional case when $p_0 < -5$ or $p_0 > 50$, the result is set to zero. Calls `pforw` in order to compute the population trajectory. Calls `evalgauss` in order to integrate `lhood`.

12. `SUBROUTINE pforw` - Computes the population size trajectory and $\frac{\partial P_T}{\partial P_0}$ for a set of parameters. In the ordinary case, this is done by using (1) and (9). In the exceptional case when $P_s < 10^{-30}$ for some s, P_t is set to 10^{-30} for $t = s, s+1, \ldots, T$. In the exceptional case when $P_0 > 2 \cdot 10^{10}$, the population size trajectory may not be accurately computed, and therefore the population size trajectory is computed in two ways. If the results are significantly different, this will be reported through the output value of the parameter `IFAIL` from the subroutine `CATCHLIMIT`.

13. `SUBROUTINE grule` - Computes the $[(n+1)/2]$ nonnegative abscissas x_i and corresponding weights w_i of the n-point Gauss-Legendre integration rule, normalized to the interval $[-1, 1]$, see Section 4.1.

14. `SUBROUTINE putgauss` - Sets up the coefficients for a n-point Gauss-Legendre integration rule for the b-integral. Calls `grule`.

15. `SUBROUTINE evalgauss` - Approximates a one-dimensional integral of a function using the Gauss-Legendre integration rule, see Section 4.1.

16. `SUBROUTINE prodgauss` - Sets up integration w.r.t. μ and p_0. This is Step 10b in the approximation procedure described in Section 3. Calls `grule` and then applies (38) to find the abscissas and the weights for the p_0-integration. The limits of the p_0-integrals are found by calling `getSplit` to get the value of $p_{0,split}(\mu)$ (defined by (15) and set by `setSplit`), and by calling `xbrent` to find $p_{0,z}(\mu)$ (defined by (27)). In this case `xbrent` finds the zero of `intLevel` for the appropriate choice of μ and D_T.

17. `SUBROUTINE halfgauss` - Sets up integration w.r.t. μ and p_0. This routine is used in Steps 5 and 8 in the approximation procedure described in Section 3. Calls `grule` to find the abscissas and the weights for the u- or v-integration, and then applies (38) to find the abscissas and the weights for the p_0-integration. The limits of the p_0-integrals are found by calling `getSplit` to get the value of $p_{0,split}(\mu)$ (defined by (15) and set by `setSplit`), and by calling `xbrent` to find $p_{0,lowmid}(\mu)$ (defined by (16)) or $p_{0,highmid}(\mu)$ (defined by (23)). In this case `xbrent` finds the zero of `logptoldt` for the appropriate choice of μ and D_T.

18. `SUBROUTINE evalpgauss` - Approximates a two-dimensional integral of a function using iterated integration and Gauss-Legendre rules (see Section 4.1) to evaluate the iterated integrals.

19. `REAL*8 FUNCTION logptoldt` - Computes $\ln(P_T) - \ln(P_0) - \ln(D_T)$. Calls `pforw` in order to compute P_T.

20. `REAL*8 FUNCTION intLevel` - Determines the internal catch limit (5) as a function of p_0. Calls `pforw` in order to compute P_T.

21. `REAL*8 FUNCTION xbrent` - Finds a zero of a function using Brent's method (see Section 4.2).

22. `REAL*8 FUNCTION zbrent` - Finds a zero of a function using Brent's method (see Section 4.2). Except for some additional parameters, this function is equal to the function `xbrent`. Two copies are needed in order to avoid recursion.

23. `REAL*8 FUNCTION getSplit` - Gets the value of $p_{0,split}(\mu)$ defined by (15) for a given μ.

24. `SUBROUTINE setSplit` - Find the abscissas and weights for the μ-integral. Determines and stores the split points ($p_{0,split}(\mu)$ defined by (15)) for each μ used as abscissa in the integration rule. Calls `grule` and then applies (38) to find the abscissas and the weights for the μ-integration. Calls `xbrent` in order to find the zero of `logptoldt` ($p_{0,split}(\mu)$).

25. `REAL*8 FUNCTION halfInt` - Calculates a semi-infinite integral, I_{lower} or I_{upper}, of the scaled likelihood (the right-hand side of (4)). Calls `halfgauss`. Calls `evalpgauss` in order to integrate `dens`.

26. `REAL*8 FUNCTION fract` - Calculates the cumulative probability of the internal catch limit at x. Subtracts the probability level α from the result. Calls `prodgauss`. Calls `evalpgauss` in order to integrate `dens`.

C Changes

Changes between versions of April 1999 and June 1999: In the version of April 1999, the variance covariance matrix of the logarithm of the abundance estimates was input. Moreover, this matrix was assumed to be diagonal and specified by a one- dimensional array containing the diagonal elements only. In the version of June 1999, however, the information matrix of the logarithm of the abundance estimates is input. This matrix does not need to be diagonal.

When `IN_INFOLEVEL` is positive, a warning message is printed if this matrix is not nonnegative definite. Due to numerical inaccuracy a singular matrix may be declared as not being nonnegative definite. In such cases, however, the magnitude of the lowest eigenvalue computed by the module is small. In order to guide the user, this eigenvalue is printed along with the warning message.

In the version of June 1999, D_1, D_2, and D_3 in (32-34) are printed if `IN_INFOLEVEL` is 6 or greater. In order to print likelihood and density values, `IN_INFOLEVEL` must be at least 7.

In the version of June 1999, `IFAIL = 6` on exit, means that the information matrix of the logarithm of the abundance estimates is not nonnegative definite.

Changes between versions of June 1999 and June 2000: In the version of June 2000, the sequence `AB_YEARS(I); I=1,...,ABDIM`; should be strictly increasing. This is checked in the subroutine `checkdat`.

In the version of June 2000, the upper bound of the interval in which the solution is seeked can be greater than in the version of June 1999. In the version of June 1999, the upper bound is $\mu_{\max}A_\tau$, where A_τ is the most recent abundance estimate. This bound could be too small if the variance of the most recent abundance estimate is large.

In the version of June 2000, the upper bound is $\mu_{\max}A_{\tau-1}$, where $A_{\tau-1}$ is the second most recent abundance estimate, provided that $A_\tau < A_{\tau-1}$, and the variance of the second most recent abundance estimate is smaller than the variance of the most recent abundance estimate. Otherwise, the upper bound is the same as in the version of June 1999.

Changes between versions of June 2000 and November 2000: The initial value of `last_quota` in `calc_quota` is changed from 0 to -10^{30} in order to avoid too early termination.

Changes between versions of November 2000 and June 2005: All real variables and constants are now in double precision. The Jacobi determinant $J(\mu, p_0, b)$, which should be a nonnegative number, is now allowed be a tiny negative number in order to avoid termination due to numerical errors.

Changes between versions of June 2005 and January 2006: Three new input parameters are added. These are `IN_PSLOPE`, `IN_PSCALE` and `IN_NOF_RULE`. Introduction of `IN_PSLOPE` and `IN_PSCALE` implies that the values of γ in (5) and s in (4) can be specified by the user. In the previous versions, γ and s were always equal to 3 and $1/16$, respectively. In addition, the maximum number of iterations allowed to compute the catch limit can now be adjusted through the parameter `IN_NOF_RULE`. In the previous version `IN_NOF_RULE` was equal to 8.

In the updated version, the upper limit of the interval in which the solution is seeked, is multiplied by two until the interval contains a solution. If we could find exact values of the integrals, we would eventually find a solution. This is because I_z is a bounded and nondecreasing function of z. However, an approximation of I_{z_1} may be larger than an approximation of I_{z_2} even if $z_1 < z_2$. If this happens, the upper limit will not be increased further, and the search terminates without finding a solution. Failing to find a solution in this case, will not be serious if the integrals can be computed with higher accuracy in subsequent iterations.