5-2003

Population Dynamics and Spatial Analysis of the Maine Green Sea Urchin (Strongylocentrotus droebachiensis) fishery

Robert C. Grabowski

POPULATION DYNAMICS AND SPATIAL ANALYSIS OF THE MAINE GREEN SEA URCHIN (STRONGYLOCENTROTUS DROEBACHIENSIS) FISHERY

By

Robert C. Grabowski

B.S. Illinois Wesleyan University, 2000

A THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

(in Marine Biology)

The Graduate School

The University of Maine

May; 2003

POPULATION DYNAMICS AND SPATIAL ANALYSIS OF THE MAINE GREEN SEA URCHIN (STRONGYLOCENTROTUS DROEBACHIENSIS) FISHERY

By Robert C. Grabowski

Thesis Advisor: Dr. Yong Chen

An Abstract of the Thesis Presented
in Partial Fulfillment of the Requirements for the
Degree of Master of Science
(in Marine Biology)
May; 2003

Fisheries research on the green sea urchin in Maine has been limited despite its importance to the state's fishing industry. The objective of this thesis was to generate critical information for the management and monitoring of the Maine green sea urchin fishery. In particular there are three main areas of interest: (1) an investigation of biological reference points; (2) spatial analysis and biomass estimation, and (3) the development of a simulation framework approach to determine an optimal sampling strategy for the fishery-independent survey program.

Biological reference points are markers commonly used to monitor and manage fisheries. For the Maine sea urchin fishery, no biological reference point had been estimated as a management target, which made it difficult to determine the status of the stock and develop appropriate management plans. The purpose of this study was to investigate if $F_{0.1}$ and F_{max} are appropriate management targets for the Maine sea urchin fishery and how uncertainties associated with them affect their suitability as management

targets. A Monte Carlo simulation approach was used with fishery-dependent data to estimate uncertainties in the biological reference points $F_{0.1}$ and F_{max}. $F_{0.1}$ was considered a more suitable as a management target than F_{max} because it is precautionary, more robust to estimation uncertainty and usually well defined. Current fishing mortality was greater than $F_{0.1}$ for all tested variations; in other words, the stock is overfished.

Estimates of exploitable biomass and current exploitation rate are essential for determining the current status of the sea urchin stock. With the onset of a fisher-independent survey program, it became possible to conduct a stock assessment that incorporates spatial variability. The objective of this study was to investigate the large-scale spatial patterns in sea urchin abundance to estimate the fishery's exploitable biomass. Triangulated irregular networks (TINs) were used to characterize the large-scale patterns in the fishery-independent density data by size category and depth. Exploitable biomass estimates were almost identical to estimates calculated using a length-structured fisheries population dynamics model on fisheries-dependent data, providing independent validation of the estimates.

The 2001 pilot study for the fishery-independent survey program was extensive, time-consuming and costly, and needed to be optimized to ensure its feasibility as a long-term scientific survey. The high degree of spatial variability in sea urchin abundance, however, prevented us from using standard optimization techniques, such as traditional statistics or even geostatistics. Kernel estimation and computer simulations were combined to create a framework for survey optimization. Optimization must decrease sampling intensity, yet produce accurate realizations of the large-scale spatial structure and be compatible with the planned statistical analysis. Considering that the sea urchin data will continue to be analyzed by traditional and spatial statistics, we chose the original fishery-independent survey with a reduction to 10 locations per strata as the optimal strategy.

The research presented in this thesis provides the DMR with essential information on the sea urchin stock, suggests new analysis techniques, and recommends a cost and time effective plan for collecting quality long-term fishery-independent data.

ACKNOWLEDGEMENTS

I would like to thank the staff at the Maine Department of Marine Resources for collecting and compiling the sea urchin fishery data. I would especially like to thank Dr. Yong Chen, Dr. Thomas Windholz, Margaret Hunter and Robert Russell from the DMR, Kathryn Wisz, lab assistant, and my friends and colleagues at the University of Maine for their assistance and invaluable advice. I would also like to thank my committee members, Dr. Robert Vadas and Dr. Les Watling. This project was partially supported by grants from the Northeast Consortium (UNH SUB 302-628), the Maine Department of Marine Resources (G1102012), and the Sea Urchin Zone Council to Y. Chen and a Maine Marine Science Fellowship from the Marine Department of Marine Resources and the University of Maine School of Marine Sciences to R. Grabowski.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

The green sea urchin, *Strongylocentrotus droebachiensis*, is a benthic echinoderm with a wide arctic-boreal distribution (Scheibling and Hatcher 2001). It is most commonly found on rocky substrate in the shallow subtidal, but can be found down to 300m in depth. The green sea urchin is generally associated with laminarian kelps, but destructive grazing can reduce the kelp beds to "barrens" dominated by crustose coralline algae. Urchin densities vary considerably by habitat; low densities of cryptic adults occur within kelp beds but high densities occur within barrens and feeding fronts.

The green sea urchin is omnivorous, however, most of its diet consists of macroalgae (Scheibling and Hatcher 2001). Laminarian kelps are the primary component of the diet and can be in the form of attached fronds, drifting fronds or detritus. Urchin aggregations occur primarily because of food availability, leading to the formation of feeding fronts (Vadas et al. 1986). Individual growth rates vary considerably due to food quality and availability, as well as sea urchin density; therefore, growth is greatest for individuals in kelp beds and lowest in urchin barrens.

The green sea urchin stores nutrients in its gonads year-round prior to the production of gametes (Scheibling and Hatcher 2001). The rich gonadal tissue is the target for the commercial fishing industry. The urchin populations in Maine reproduce and spawn once a year, between February and May (Vadas et al. 1997). There are spatial and temporal differences in spawning within the Gulf of Maine. Western populations spawn 4 to 6 weeks earlier than eastern ones, whereas, eastern populations spawn for 4 to

6 weeks longer than western ones. Fertilization is external and the planktotrophic larvae can remain in the water column for 4 to 21 weeks (Scheibling and Hatcher 2001). There are large-scale recruitment patterns within the Gulf of Maine that correspond to coastal circulation patterns and depth. At smaller scales, recruitment is spatially and temporally variable and appears to be lower in kelp beds than in barrens. Post-settlement mortality rates are higher in fleshy macroalgal habitats, possibly due to higher densities of micropredators (McNaught and Steneck 1998)

The green sea urchin can profoundly alter the rocky subtidal habitat. Destructive grazing can transform lush, diverse kelp beds into urchin barrens (Scheibling and Hatcher 2001). High levels of sea urchin recruitment and low mortality rates, along with high levels of algal predation, help to maintain the urchin barren state. Conversely, high levels of algal recruitment, along with low levels of sea urchin recruitment and high urchin mortality rates, help to maintain the kelp bed state. For these reasons, kelp beds and urchin barrens have been considered alternate stable states, and may be locally stable for decades (Vavrinec and McNaught 2001).

Urchin barrens are currently targeted for the sea urchin fishery; however, these barrens may be an artifact of the historic fishing industry in Maine. Research has suggested that an anthropogenic release in predation on sea urchins, due to intensive fishing on lobster and predatory fishery, may have allowed the development of high-density feeding fronts and barrens (Vadas and Steneck 1995). The large-scale removal of urchins through the commercial fishing industry could feasibly shift urchin barrens back to kelp beds (Vavrinec and McNaught 2001). In fact, researchers have documented this type of habitat switch in eastern portions of the Gulf of Maine, which historically received the earliest and heaviest fishing pressure.

The green sea urchin is an important component of the fishing industry in the state of Maine, currently ranking fourth by value. Commercial landings began in the late 1980's, quickly reaching a peak of more than 22,000 metric tons in 1993 (Figure 1.1.). It has since experienced a continuous declining trend in yield, with landings of less than 5000 metric tons in 2001. This decline has been attributed to decreasing stock abundance over the last decade (Chen and Hunter 2003).

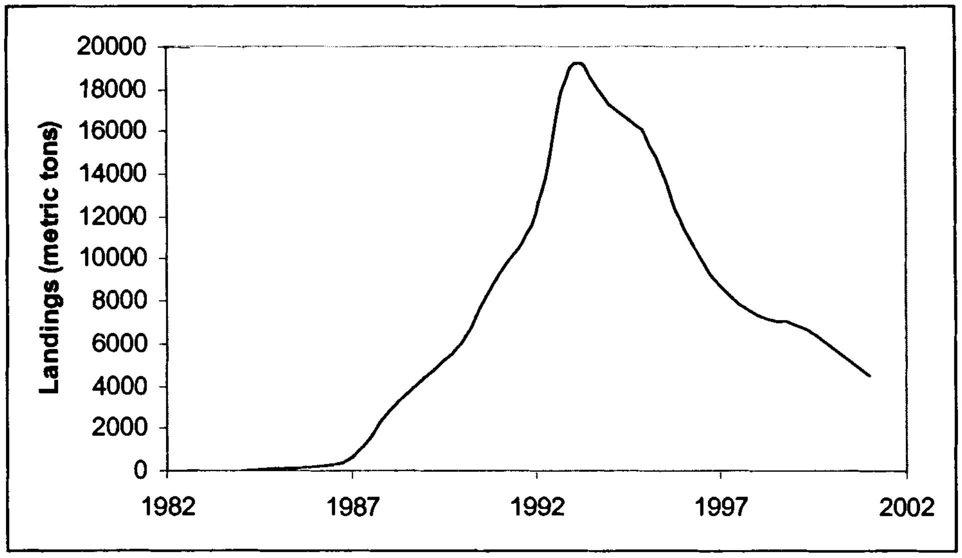


Figure 1.1. Commercial landings in metric tons for the Maine sea urchin fishery.

The sea urchin fishery is managed by the Maine Department of Marine Resources (DMR) using a number of management tools, including limited entry, opportunity days and minimum (52mm) and maximum (76mm) size limits. The fishing grounds are divided into two management zones based on spatial/temporal variations in spawning, in which management differs only by fishing seasons (Vadas et al. 2002) (Figure 1.2.). Fishery-dependent data, including landings and catch size compositions, have been collected by the DMR since the commercial fishery started. This information has formed the basis of subsequent management decisions. A fisheries-independent survey program commenced in 2001 and provides annual estimates of abundance, spatial distribution and population structure for the green sea urchin stock along the coast of Maine.

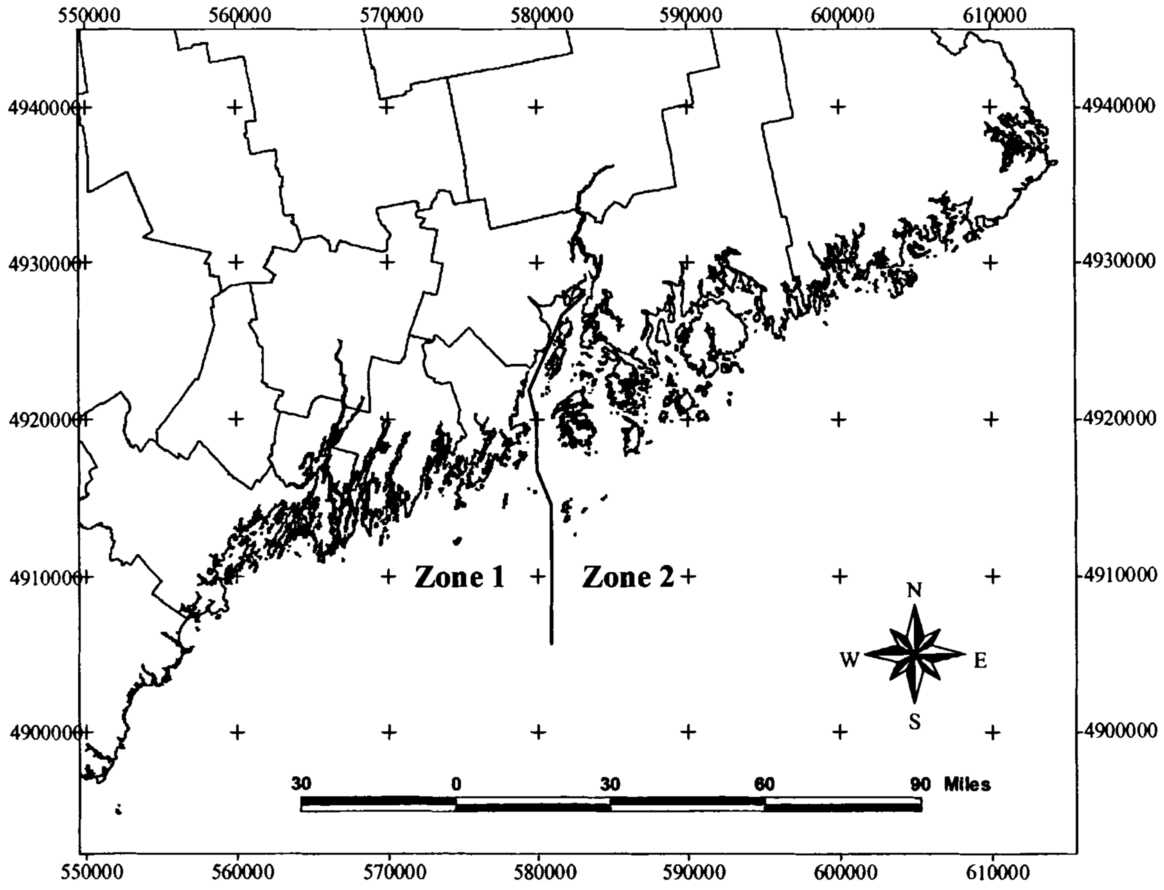


Figure 1.2. Management zones for the Maine sea urchin fishery.

The objective of this research is to utilize the fishery-dependent data and the fishery-independent survey to generate critical information for the management and monitoring of the Maine green sea urchin fishery. In particular there are three main areas of interest: (1) an investigation of biological reference points, (2) spatial analysis and biomass estimation, and (3) development of a simulation framework approach to determine an optimal sampling strategy for the fishery-independent survey program. A determination of the sea urchin stock status demands an estimate of the current stock biomass as well as target fishing rates. The best way to gather this information is from a carefully designed fishery-independent survey program. However, if the annual survey places an unrealistic burden of time and cost on fishery managers, it is less likely to continue over the years. Fisheries research on the green sea urchin in Maine has been limited despite its importance to the state's fishing industry. This research is of critical importance to the DMR because it provides essential information on the sea urchin stock, as well as a cost and time effective plan for collecting quality long-term fishery-independent data.

Chapter 2

INCORPORATING UNCERTAINTY INTO THE DEVELOPMENT OF BIOLOGICAL REFERENCE POINTS $F_{0.1}$ and F_{max}

Chapter Abstract

The first formal stock assessment of the Maine green sea urchin fishery was conducted in 2001; however no biological reference points have been developed to monitor and manage the fishery. The purpose of this study is to investigate if $F_{0.1}$ and F_{max} are appropriate management targets for the Maine sea urchin fishery and how uncertainties associated with them affect their suitability as management targets. A yield per recruit model is developed with parameters derived in the 2001 stock assessment to estimate F_{max} and $F_{0.1}$. Using a Monte Carlo simulation approach, we estimate uncertainties of $F_{0.1}$ and F_{max}. This study suggests that $F_{0.1}$ is more suitable as a management target than F_{max} for the Maine sea urchin fishery because it is precautionary, more robust to estimation uncertainty and usually well defined. Since different biological reference points indicate different aspects of the fish stock structure and may result in different conclusions on the fishery status, we suggest that further investigations into other reference points be conducted before final selection and implementation of a management target for the Maine sea urchin fishery.

Introduction

The first formal stock assessment of the Maine green sea urchin fishery was conducted in 2001 (Chen and Hunter 2003). This study used fishery data and urchin life history parameters estimated independently from scientific studies to assess the population dynamics of the Maine urchin stock. A robust Bayesian approach was used with a length-based stock assessment model to determine current and historical stock biomasses and exploitation rates. Determining the current status of the Maine urchin stock, however, requires a comparison of the current stock biomass and/or exploitation rate estimated in the stock assessment with a biological reference point.

Biological reference points are markers commonly used to monitor and manage fisheries (Hilborn and Walters 1992). They perform three main functions: (1) management target; (2) management threshold; and (3) management limit. A management target is a reference point that is aimed for, and can be either a rate, such as fishing mortality, or an absolute target, such as spawning stock biomass. Threshold and limit reference points represent maximums that if reached necessitate action to enhance the stock and rebuild stocks from over-exploitation (Jennings et al. 2001).

For the Maine sea urchin fishery, although the current stock abundance and exploitation rate were derived in the 2001 stock assessment (Chen and Hunter 2003), no biological reference point had been estimated as a management target, which made it difficult to determine the status of the stock and develop appropriate management plans.

For a given fishery, biological reference points can be determined through a variety of means, such as a yield-per-recruit (YPR) analysis, a stock-recruitment model, or a production model (Jennings et al. 2001). Different biological reference points represent different aspects of the fish stock structure and may result in different conclusions on the fishery status. $F_{0.1}$ and F_{max} are biological reference points derived from YPR models. F_{max} corresponds to the fishing mortality that results in the highest

YPR and is often considered a management target (Ricker 1975). $F_{0.1}$ is always less than F_{max}, and corresponds to a fishing mortality beyond which increases in fishing effort produce marginal increases in YPR values (Rivard and Maguire 1993). Even though the designation of $F_{0.1}$ was not based on theoretical grounds, there are a number of benefits of using $F_{0.1}$ as a management target instead of higher fishing mortalities such as F_{max} (Deriso 1987). Since $F_{0.1}$ corresponds to a lower YPR value, overexploitation of the stock is less likely to occur when a YPR model is not well parameterized. Thus using $F_{0.1}$ as a management target is consistent with the precautionary management strategy proposed by FAO (Food and Agriculture Organization 1995). $F_{0.1}$ usually can be well defined, whereas F_{max}, on the other hand, can be difficult to define or even indefinable. Finally, $F_{0.1}$ is less sensitive than F_{max} to changes in model parameters, especially when YPR curves have a poorly defined F_{max} (Rivard and Maguire 1993). $F_{0.1}$ is regarded as a more appropriate management target (Mace 1994), compared with F_{max}, and has been widely used in management of many fisheries in the world (Quinn and Deriso 1999; Restrepo 1999). In this study we evaluate the status of Maine sea urchin stock by comparing current fishing mortality in the urchin fishery with $F_{0.1}$ and F_{max} as biological reference points and discuss the suitability of using $F_{0.1}$ and F_{max} as a management target in managing the Maine sea urchin stock.

Regardless of which biological reference point is chosen, there are uncertainties associated with the process and model that can have large effects on the estimation of biological reference points, and subsequently on the status assessment of a fish stock (Helser et al. 2001; Chen and Wilson 2002). The importance of incorporating uncertainties in fisheries advice has been well noted in recent literature (e.g. Hilborn and

Walters 1992; Restrepo 1999; Mace 2001). Uncertainty originates from various sources. Francis and Shotton (1997) reviewed the literature and summarized six types of uncertainties: process, observation, model, estimation, implementation and institutions, of which, the first four are most relevant to this study. Process uncertainty arises from natural variability in populations, such as annual variations in recruitment. Observation uncertainties occur because of measurement and sampling errors associated with sampling programs. Model uncertainty arises from the "lack of complete information on the population and community dynamics of the system" (Fogarty et al. 1996). Estimation uncertainty is caused by the choice of statistical approach used for parameter estimation (Francis and Shotton 1997). In this study we evaluate how uncertainties in parameters of the YPR model may affect the estimation of $F_{0.1}$ and F_{max} for the Maine sea urchin fishery.

The inclusion of uncertainties in stock assessment is important to prevent erroneous conclusions about the status of the fish stock (Smith et al. 1993). However, few studies have investigated how uncertainty in both current fishing mortality/biomass and biological reference points may affect the assessment of stock status. Helser et al. (2001) and Chen and Wilson (2002) demonstrated that uncertainties in both current fishing mortality and biological reference points could have a large impact on the status assessment of a fishery. High uncertainty makes it more difficult to conclude that a stock is overfished (Chen and Wilson 2002).

In this study, we estimate $F_{0.1}$ and F_{max} and evaluate the uncertainty associated with them for the Maine sea urchin stock, and then compare them with fishing mortality rate and its associated uncertainty estimated in the 2001 sea urchin stock assessment.

Such a study will provide us with insights on suitability of using $F_{0.1}$ and F_{max} as management targets for the Maine sea urchin stock, and help gauge the current status of the Maine sea urchin stock.

Materials and Methods

Estimation of $F_{0.1}$ and F_{max} and their associated uncertainties

The commonly used discrete YPR model can be written as

(1) $$Y = \sum_{t=t_R}^{t_\lambda} C_t W_t (1 - D_t)$$

where Y is the attained yield, t_R is the age of entry into the fishery, and t_λ is the maximum age of fish that still contribute to the fishery. D_t is the proportion of fish caught at age t that are discarded at sea. W_t is the weight of fish at age *t*, and is commonly calculated as

(2) $$W_t = aL_t^b$$

where *a* and *b* are two parameters to be estimated. L_t is the length at age t, and is often related to age through a growth function. In many studies, the relationship between length and age is described by the von Bertalanffy growth function,

(3) $$L_t = L_\infty (1 - e^{-K(t-t_0)}),$$

where L_∞ is defined as the average asymptotic length an organism may attain, K is the Brody growth parameter describing how fast organisms approach to L_∞, and t_0 is a hypothetic age at length of 0 (Ricker 1975).

Given mortality rates and selectivity coefficients, catch-at-age C_t can be calculated from the catch equation as

(4) $$C_t = N_t \frac{S_t F}{S_t F + M} (1 - e^{-S_t F - M})$$

where S_t is the selectivity coefficient for fish of age t, F is the fishing mortality for fully recruited fish, M is the natural mortality rate, and N_t is the number of fish still alive at the beginning of age t. The relationship between recruit R (i.e. number of fish at the beginning of age t_R) and N_t is given by

$$N_t = R e^{-\sum_{j=t_R}^{t-1}(S_j F + M)} \tag{5}$$

Thus, the attained YPR value can be calculated from equations (1) to (5) as

$$\frac{Y}{R} = \sum_{t=t_R}^{t_\lambda} \left[\frac{a\left[L_\infty(1-e^{-K(t-t_0)})\right]^b S_t F}{S_t F + M} (1 - e^{-S_t F - M}) e^{-\sum_{j=t_R}^{t-1}(S_j F + M)} (1 - D_t) \right] \tag{6}$$

Parameters L_∞, K, t_0, a, and b in equation (6), obtained from the 2001 sea urchin stock assessment (Chen and Hunter 2003), are 100, 0.1006, 1.0019, and 2.6234, respectively.

The selectivity and discarding parameters were set with knife-edges at the minimum legal size limit, with the selectivity at 100% and discarding at 0% once urchins have recruited to the fishery. Natural mortality, M, was set at 0.147, which is consistent with natural mortality estimated for the Maine sea urchin in management zone 1.

F_{max} is estimated by taking the derivative of Y/R with respect to fishing mortality in equation (6) as.

$$\left. \frac{d\left(\frac{Y}{R}\right)}{dF} \right|_{F\max} = 0 \tag{7}$$

By definition, $F_{0.1}$ is the instantaneous rate of fishing mortality where the slope of the yield per recruit F is 10% of the maximum slope which is at F=0. Thus, it can be estimated from the following equation:

(8) $$(0.1)\left.\frac{d\left(\frac{Y}{R}\right)}{dF}\right|_{F=0} = \left.\frac{d\left(\frac{Y}{R}\right)}{dF}\right|_{F_{0.1}}$$

F_{max} and $F_{0.1}$ estimated from equations (7) and (8) are deterministic estimates. To estimate uncertainties associated with them, we used a Monte Carlo simulation approach described in Chen (1996) to incorporate variability in parameters of the YPR model. This approach involves simulating a large number (say N) of size-at-age and weight-length data with a given level of variation, using these simulated data in equations (2) and (3) to estimate N sets of parameters in equations (2) and (3), and then using the N sets of simulated parameters in equation (6) to estimate N sets of F_{max} and $F_{0.1}$. Detailed description is described in Chen (1996) and Chen and Wilson (2002).

In this study three scenarios were simulated with varying levels of variation in simulating the length at age and weight at age data. For the base scenario (II), the standard deviation for length at age data was set at 0.25, and 0.15 for the weight at age values, which we considered to be a medium level of uncertainty (Table 2.1.). The other two scenarios, high variation (III) and low variation (I), have higher and lower variation than the base scenario, respectively. For each scenario, 300 sets of observed lengths and weights were simulated for each age category in the analysis.

The parameters in the von Bertalanffy growth equation and the length-weight relationship were then estimated using non-linear estimation. Parameters are fit to each set of observed length at age and weight at length data. A nonlinear least squares (NLS) analysis was conducted to estimate parameters in equations (2) and (3). If the estimation was converged in the NLS, the program outputted the parameter estimates and repeated the process on the next set of length at age and weight at length data. If the estimation did not converge, the NLS skipped to the next set of data.

Table 2.1. Summary of the scenarios incorporating variation into the fishery data for the simulation study.

	Scenario I Low Variation	Scenario II Medium Variation	Scenario II High Variation
STD for length at age values	0.15	0.25	0.40
STD for weight at age values	0.10	0.15	0.20

Variability in natural mortality M was found to follow lognormal distribution as $M = \overline{M}e^{\varepsilon}$, where $\varepsilon \subset N(0, \sigma^2)$. $\overline{M}$ is the mean value (i.e. 0.147) estimated in the stock assessment (Chen and Hunter 2003), and ε is the error term following normal distribution with a mean of 0 and standard deviation of σ, which was estimated to be 0.1 in the stock assessment. Three hundred simulation runs were conducted and for each run F_{max} and $F_{0.1}$ were estimated using equations (7) and (8) The probability distributions of F_{max} and $F_{0.1}$ were estimated from the resultant estimates. The procedure is graphically represented in a flowchart diagram (Figure 2.1.).

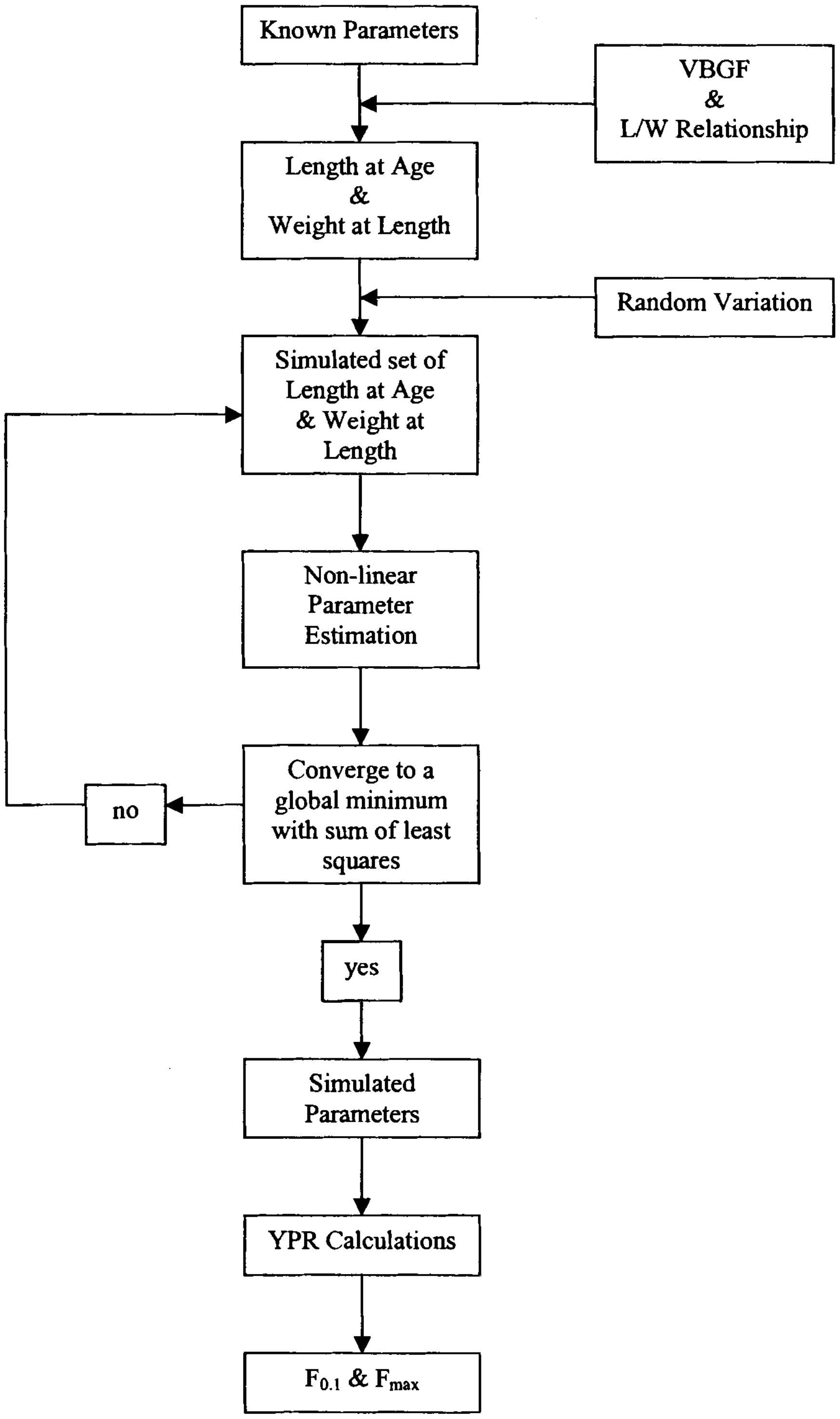
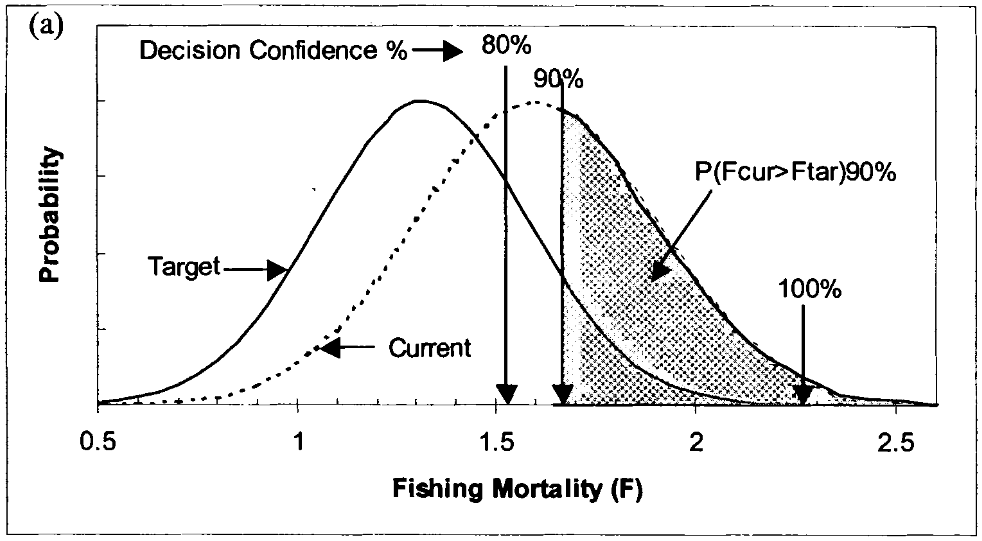
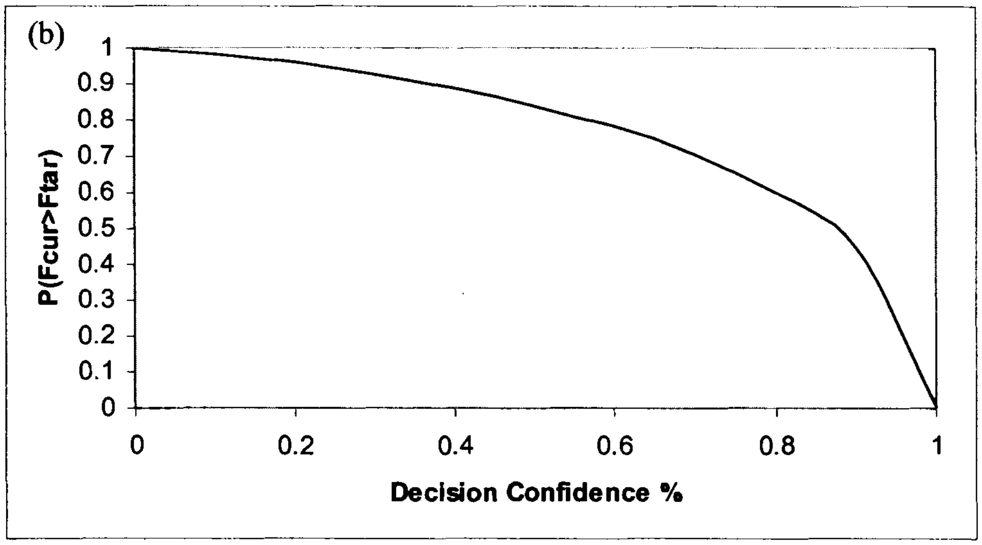

Known Parameters
VBGF & L/W Relationship
Length at Age & Weight at Length
Random Variation
Simulated set of Length at Age & Weight at Length
Non-linear Parameter Estimation
Converge to a global minimum with sum of least squares
no
yes
Simulated Parameters
YPR Calculations
$F_{0.1}$ & F_{max}

Figure 2.1. Flowchart diagram of the simulation method used to calculate probabilistic estimates of the biological reference points $F_{0.1}$ and F_{max}.

Comparison of the current fishing mortality with F_{max} and $F_{0.1}$

An empirical distribution of current exploitation rate for the Maine sea urchin fishery was estimated in the 2001 stock assessment (Chen and Hunter 2003). To determine if the stock was being over-fished, traditional methods would simply compare the current fishing mortality (F_{cur}) with a biological reference point, such as $F_{0.1}$. However, to incorporate uncertainty in both current fishing mortality and biological reference point in the decision-making process, we must compare their empirical distributions. One method to incorporate uncertainty is to determine the probability that $F_{cur}>F_{0.1}$ at a given decision confidence level (Helser et al. 2001; Chen and Wilson 2002) (Figure 2.1). The decision confidence level is synonymous with the one tailed probability of the empirical probability distribution for $F_{0.1}$. Therefore, a decision confidence level of 90% corresponds to an F-value at which 90% of the area under the $F_{0.1}$ (Target) probability distribution is to the left of that value (Figure 2.2.a.). We can then determine the probability that F_{cur} is greater than that F-value by tallying the area under the F_{cur} distribution that is to the right of the F-value. Detailed description on this approach was described in Helser et al. (2001) and Chen and Wilson (2002). Since each decision confidence level corresponds to a probability, $P(F_{cur}>F_{0.1})$, we can generate a probability profile (Figure 2.2.b.). The probability profile provides fisheries managers with a means to assess current stock status inclusive of uncertainties in both the indicator (i.e. F_{cur}) and management reference points.


(a)
Decision Confidence %
80%
90%
P(Fcur>Ftar)90%
100%
Target
Current
Probability
0.5
1
1.5
2
2.5
Fishing Mortality (F)


(b)
P(Fcur>Ftar)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
Decision Confidence %

Results

From the 2001 stock assessment, the median estimate of current exploitation rate for the Maine sea urchin fishery was calculated at 0.371 (Chen and Hunter 2003). The exploitation rate was converted into an instantaneous fishing mortality rate, with a probability distribution ranging from 0.20 to 1.30 and a median at 0.46 (Table 2.2.). This rate can be compared with estimates of biological reference points to gauge the stock status.

Table 2.2. Summary statistics for $F_{0.1}$ and F_{max} estimated in the simulation study and for the F_{cur} obtained from the urchin stock assessment (Chen and Hunter 2003).

	Scenario	Min	Max	Mean	Median	CV	Count
$F_{0.1}$	1-low variation	0.121	0.235	0.161	0.160	0.118	290
	2-medium variation	0.126	0.328	0.169	0.166	0.148	210
	3-high variation	0.119	0.230	0.161	0.159	0.118	236
F_{max}	1-low variation	0.154	0.993	0.465	0.442	0.305	290
	2-medium variation	0.148	1.354	0.445	0.428	0.369	210
	3-high variation	0.152	1.018	0.460	0.443	0.315	236
F_{cur}		0.197	1.29	0.486	0.464	0.263	2495

A YPR analysis was conducted and produced deterministic estimates of F_{max} and $F_{0.1}$ at 0.447 and 0.148, respectively (Figure 2.3.).

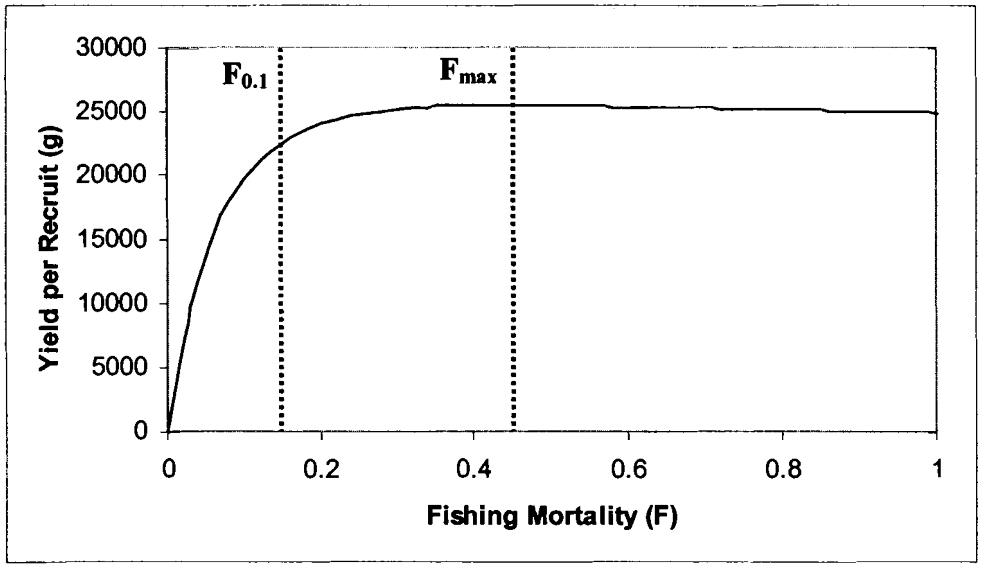


Figure 2.3. Yield per recruit vs. fishing mortality with no variations incorporated in the length-at-age and weight-at-length data, the deterministic estimates of F_{max} and $F_{0.1}$ are 0.447 and 0.148, respectively.

The three scenarios simulated to determine probabilistic estimates were identical except for the variation incorporated into the length at age and weight at length data. The probability distributions for the estimated von Bertalanffy and weight/length parameters illustrate these incorporated variations (Figures 2.4, 2.5. and 2.6.). Scenario II represents medium variation and had a $F_{0.1}$ probability distribution ranging from 0.13 to 0.33 with the median at 0.17 (Table 2.2.). $F_{0.1}$ and F_{cur} distributions were substantially different with only a slight overlap (Figure 2.7.). Since the distributions differ greatly, there is little variation in $P(F_{cur}>F_{0.1})$ over most confidence levels. $P(F_{cur}>F_{0.1})$ remained 1.00 for all confidence levels up to a 100% decision confidence level, where it dipped down to 0.94. The F_{max} distribution for scenario II was very similar to the F_{cur} distribution, ranging from 0.15 to 1.35, with a median at 0.43 (Table 2.2. and Figure 2.8.). The probability profile suggests that the probability that current fishing mortality is greater than F_{max} varies considerably with confidence levels. For example, at the 50% decision confidence level, $P(F_{cur}>F_{max})$ was 0.64, but as confidence level increased to 100%, the probability decreases to 0.

$F_{0.1}$ and F_{max} probability distributions were similar for all three scenarios, with comparable means and medians, which were also similar to the deterministic estimates (Table 2.2.). The range of values and the presence of outliers did vary with scenario. Probability distribution for the high and low variation scenarios were similar to the base scenario in that the distribution of current fishing mortality is distinctly different than $F_{0.1}$. In scenario I, the lower variation in length at age and weight at length values resulted in less variation in $F_{0.1}$ and F_{max}. The $F_{0.1}$ probability distribution did not overlap with F_{cur} (Table 2.2. and Figure 2.9.). Since F_{cur} was always greater than $F_{0.1}$, $P(F_{cur}>F_{0.1})$ remained at 1.00 for all decision confidence levels. Similar results were obtained for scenario III, despite a higher variation in the YPR analysis. The $F_{0.1}$ probability distribution for scenario III had a lower standard deviation and smaller range than scenario II (Table 2.2 and Figure 2.10.). Probability distributions of F_{max} for the low and high variation scenarios were similar to Scenario II (Figures 2.8., 2.11 and 2.12.). Summary statistics for the F_{max} distribution for all three scenarios were very similar, and overlapped considerably with F_{cur}. This translated into similar probability profiles, in which $P(F_{cur}>F_{max})$ varies with decision confidence levels, from 1.00 at 0% down to 0 at 100% decision confidence level reaches (Figures 2.7., 2.8. and 2.9.).

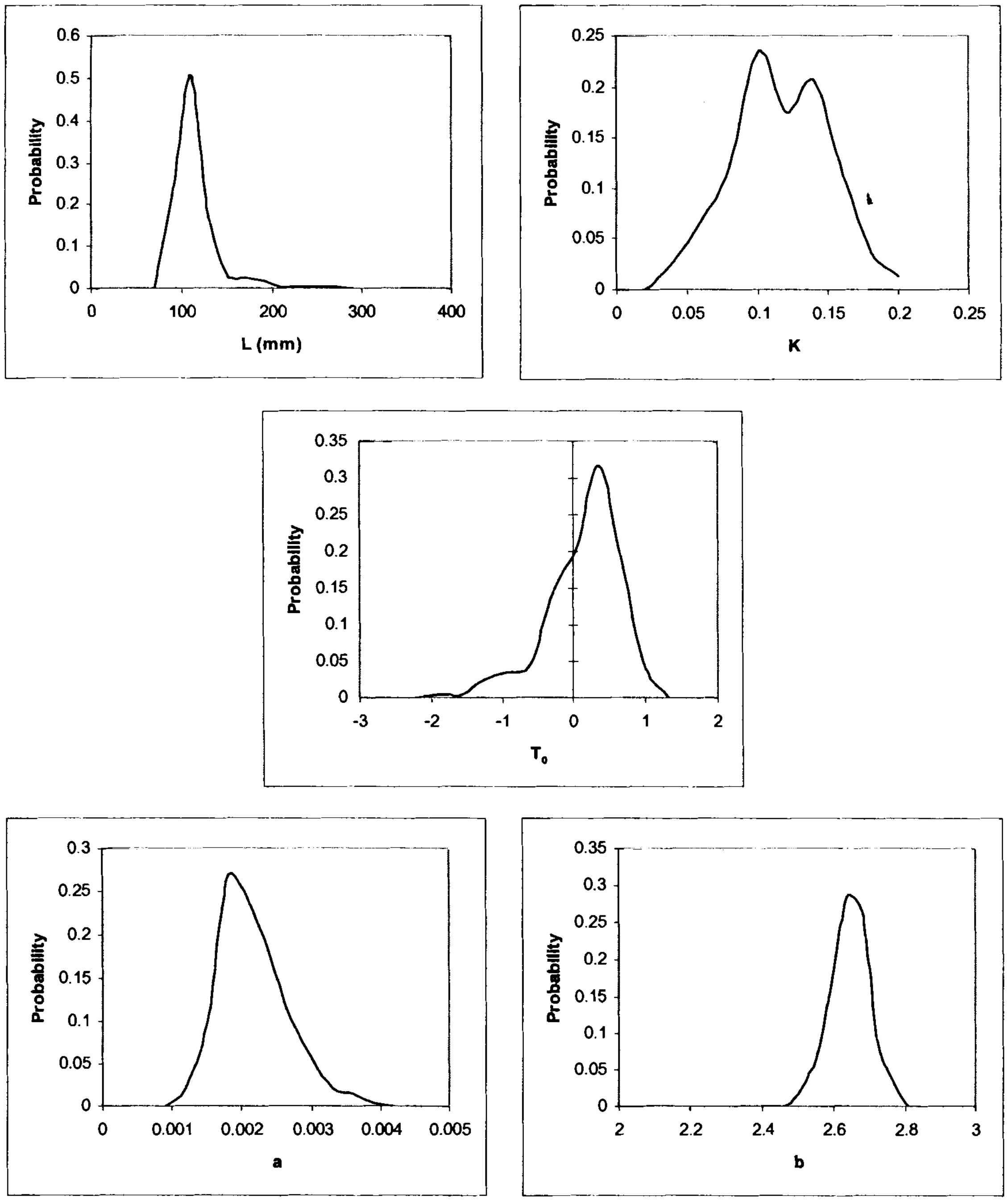


Figure 2.4. Probability distributions of the estimated parameters used in the low-variation scenario (I): Top and middle rows, estimated VBGF parameters; bottom row, estimated weight/length parameters.

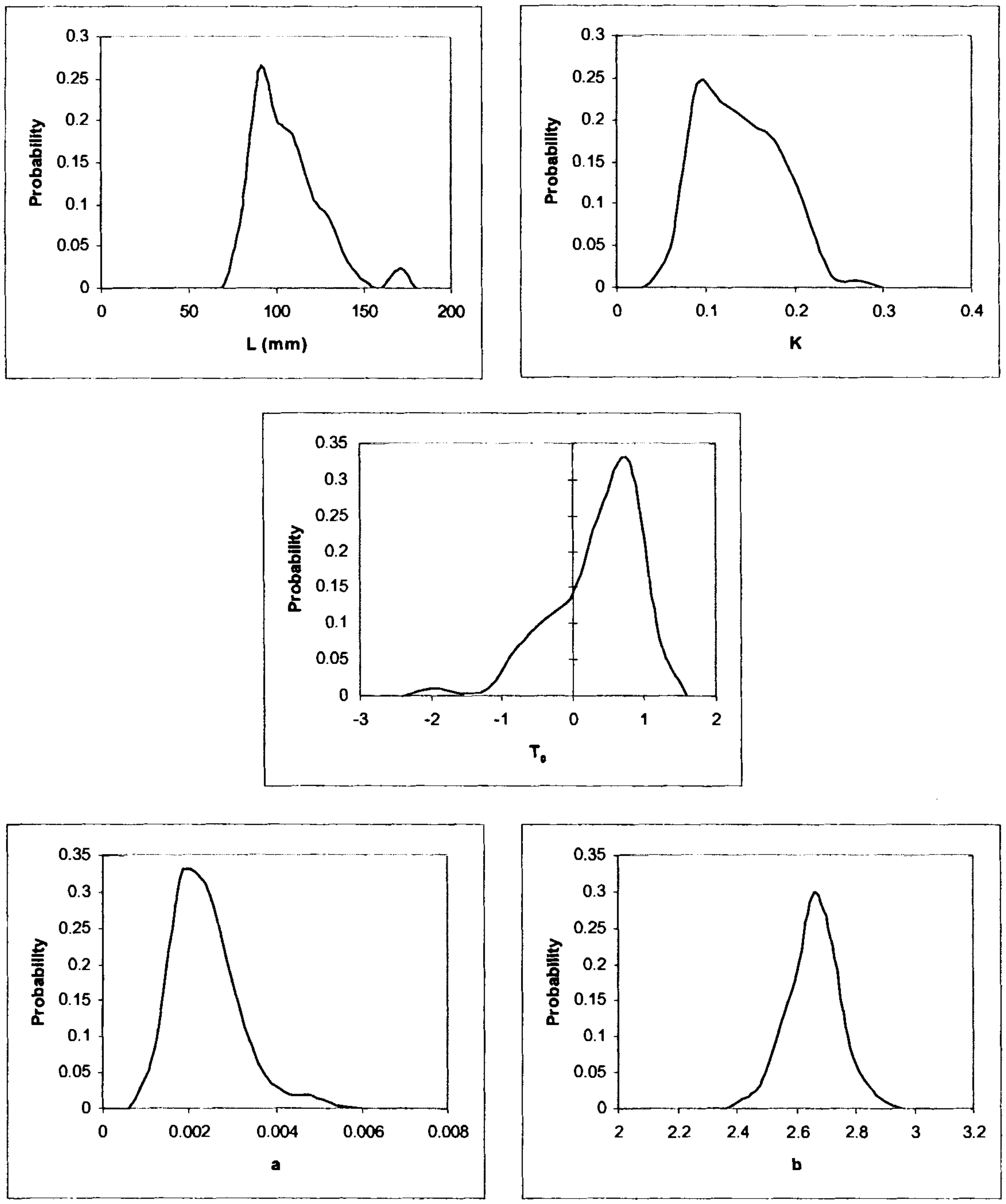


Figure 2.5. Probability distributions of the estimated parameters used in the medium-variation scenario (II): Top and middle rows, estimated VBGF parameters; bottom row, estimated weight/length parameters.

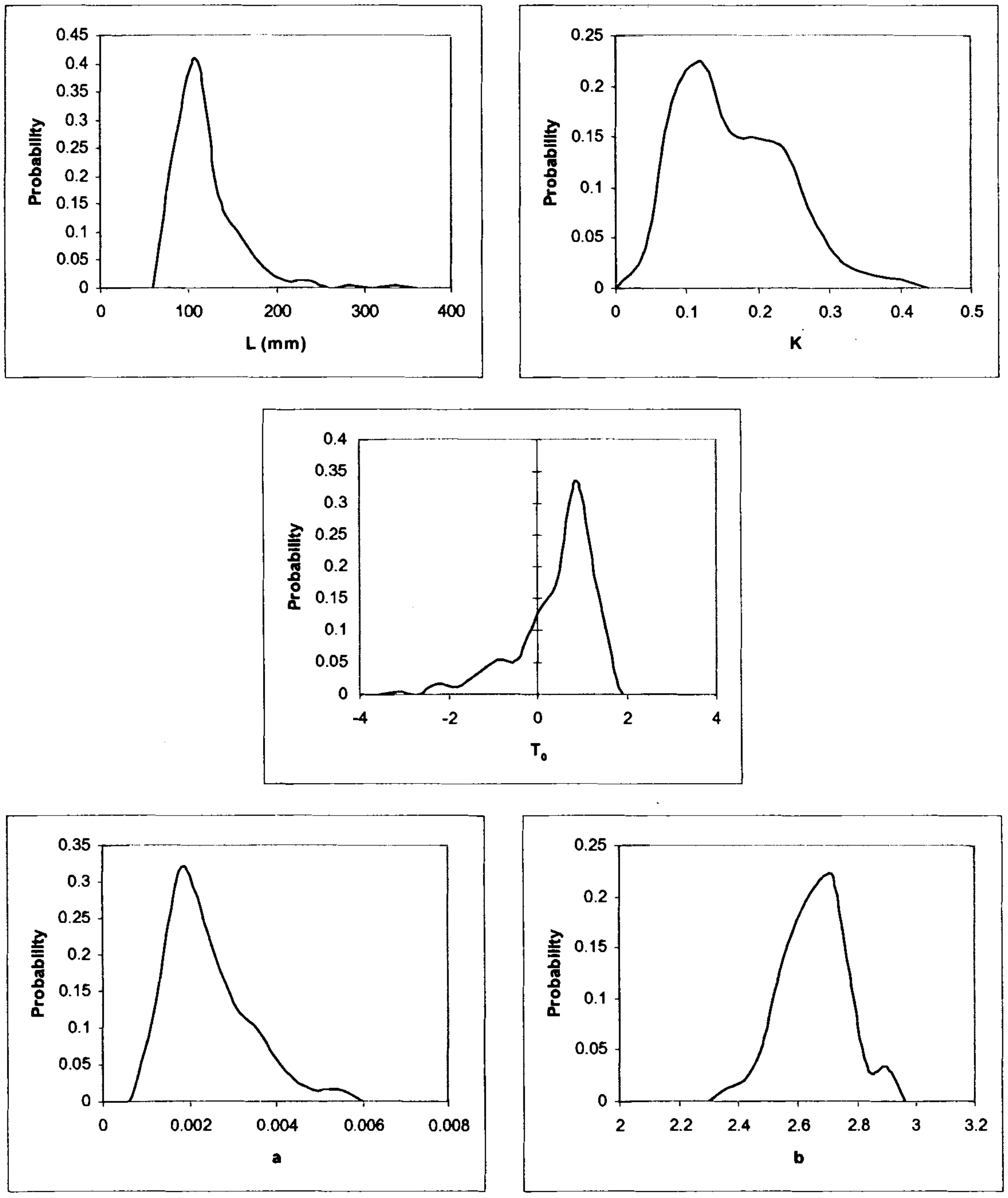


Figure 2.6. Probability distributions of the estimated parameters used in the high-variation scenario (III): Top and middle rows, estimated VBGF parameters; bottom row, estimated weight/length parameters

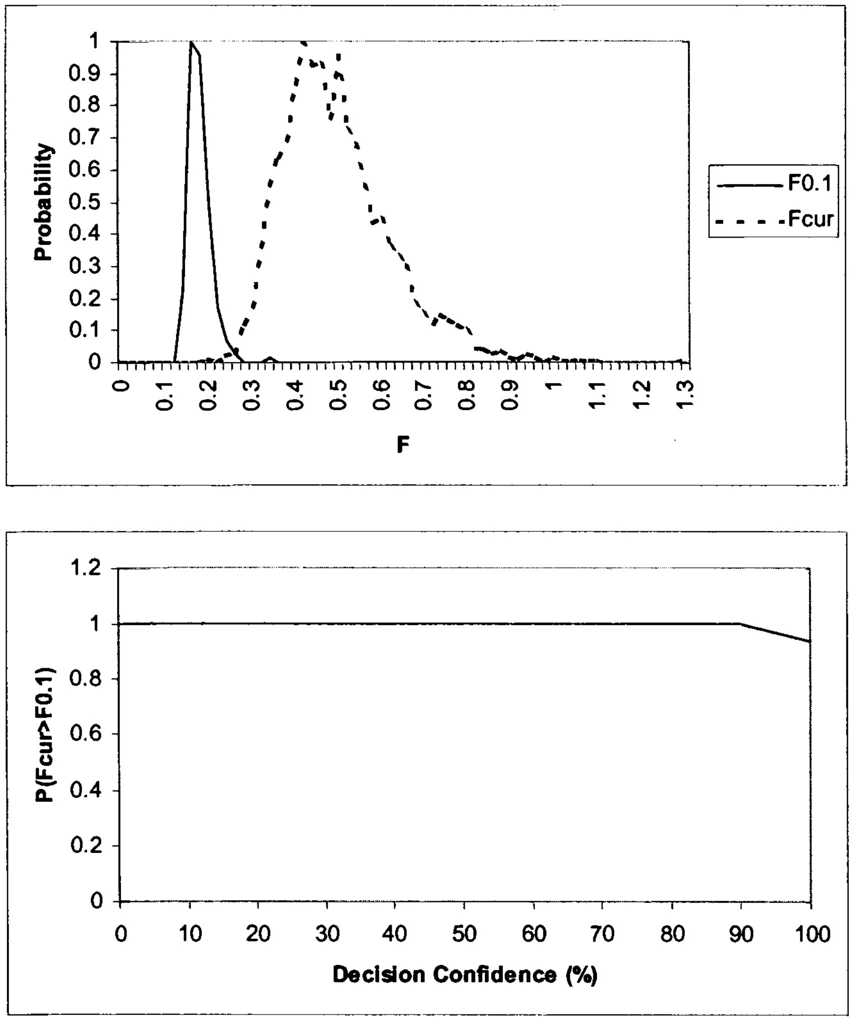
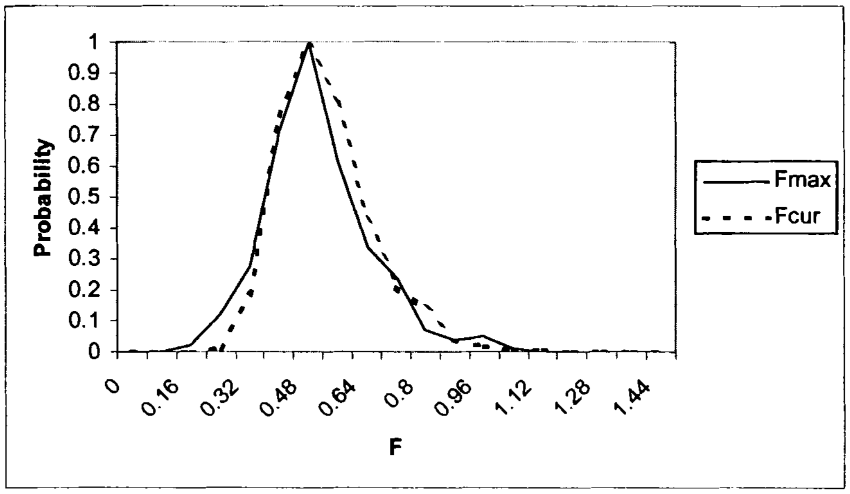


Figure 2.7. Simulation summary of the probabilistic estimate of $F_{0.1}$ for the medium-variation scenario (II): Top panel, probability distribution of $F_{0.1}$; bottom panel, probability profile specifying the $P(F_{cur}>F_{0.1})$ for varying decision confidence levels.



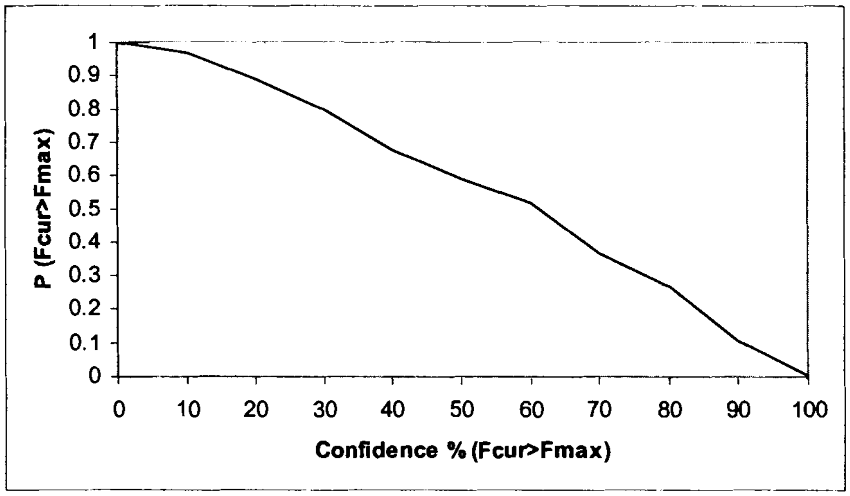
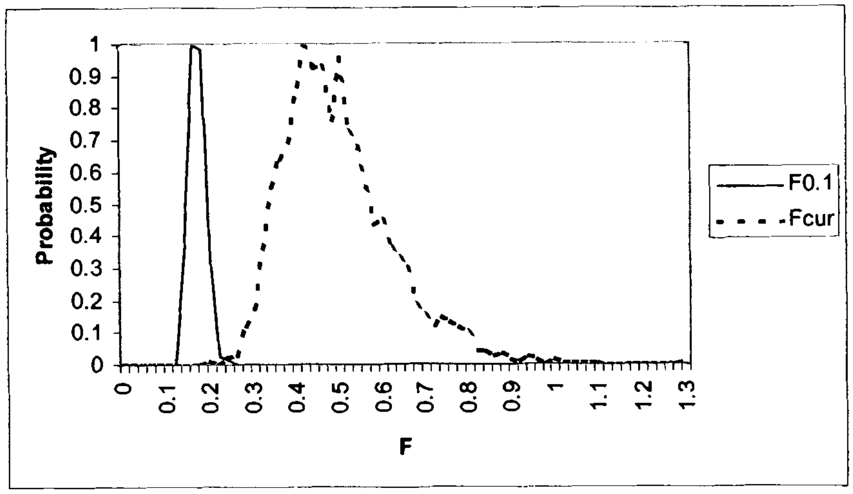


Figure 2.8. Simulation summary of the probabilistic estimate of F_{max} for the medium-variation scenario (II). Top panel, probability distribution of Fmax; bottom panel, probability profile specifying the $P(F_{cur}>F_{max})$ for varying decision confidence levels.



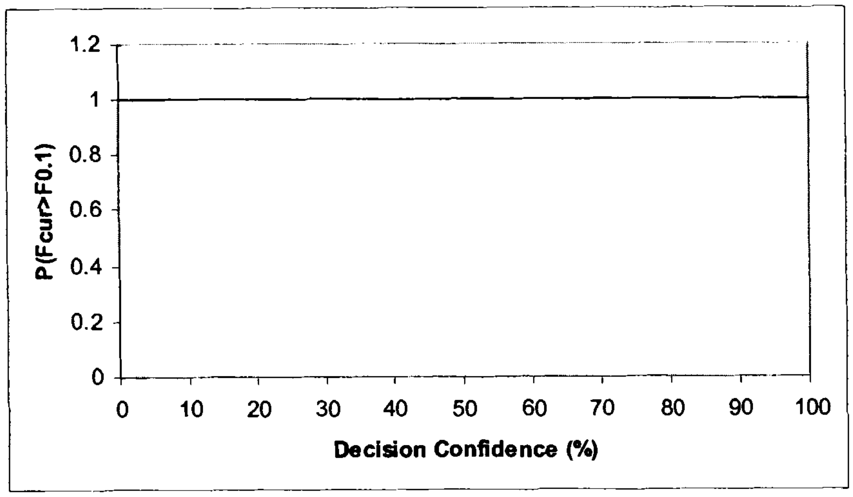
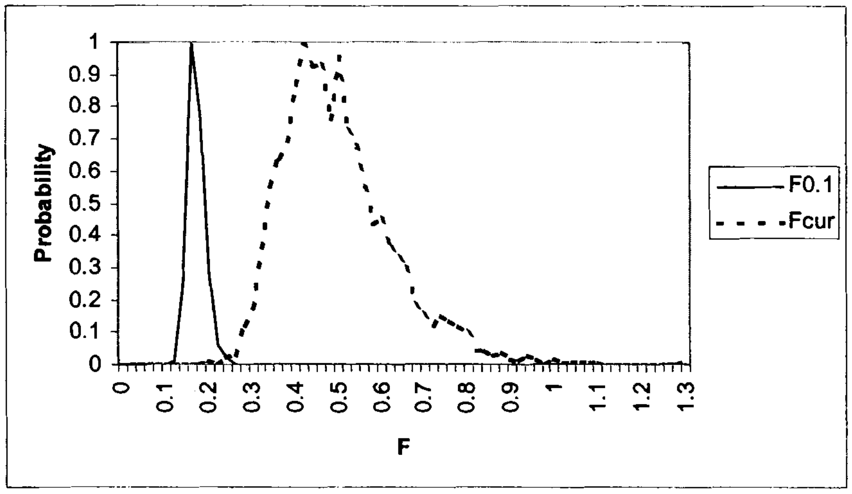


Figure 2.9. Simulation summary of the probabilistic estimate of $F_{0.1}$ for the low-variation scenario (I). The layout is the same as in Figure 2.7.



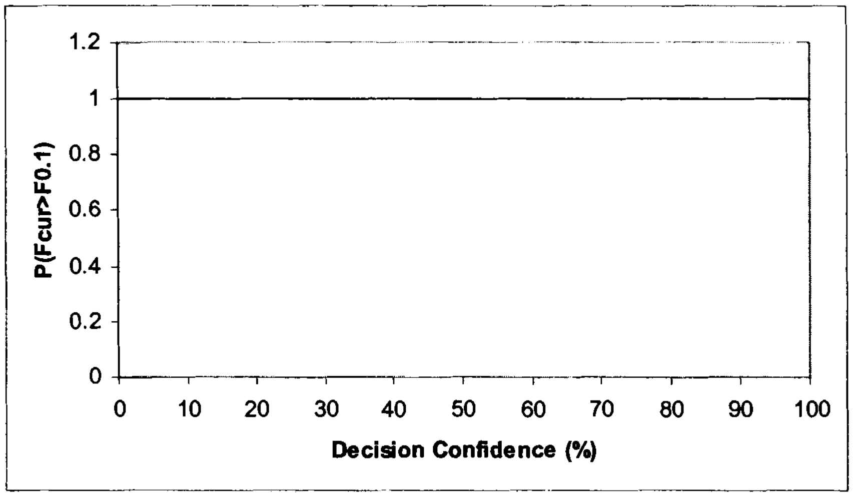
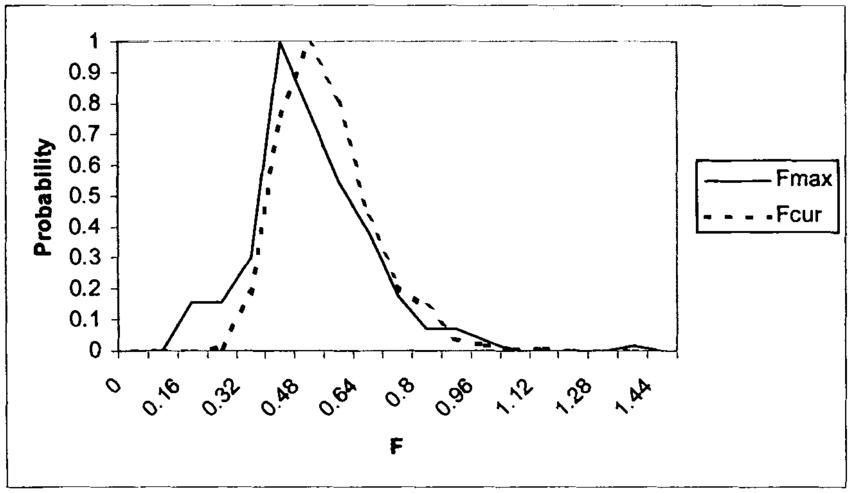


Figure 2.10. Simulation summary of the probabilistic estimate of $F_{0.1}$ for the high-variation scenario (III). The layout is the same as in Figure 2.7.



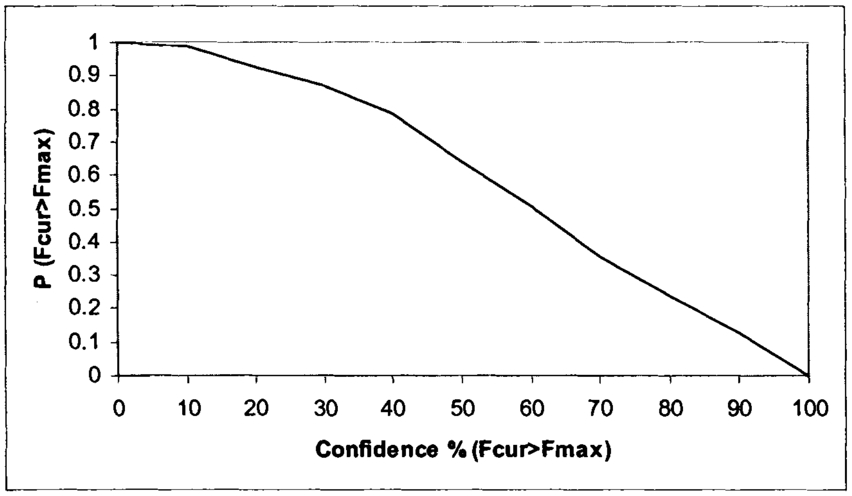
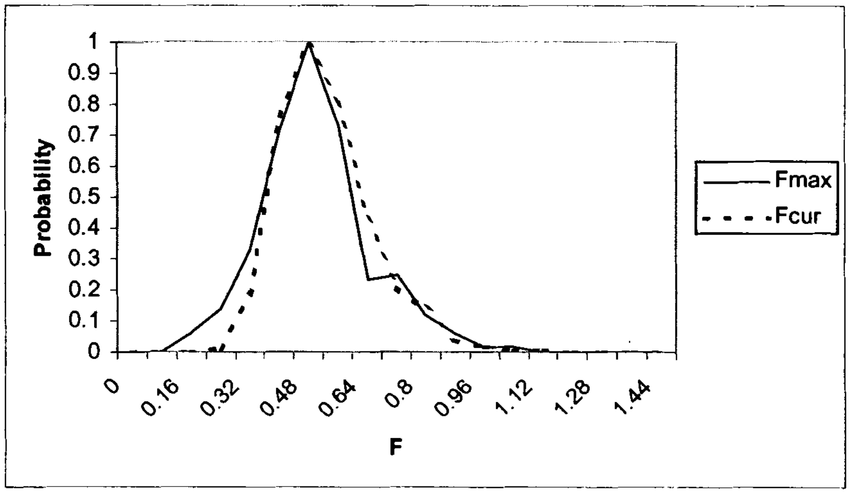


Figure 2.11. Simulation summary of the probabilistic estimate of F_{max} for the low-variation scenario (I). The layout is the same as in Figure 2.8.



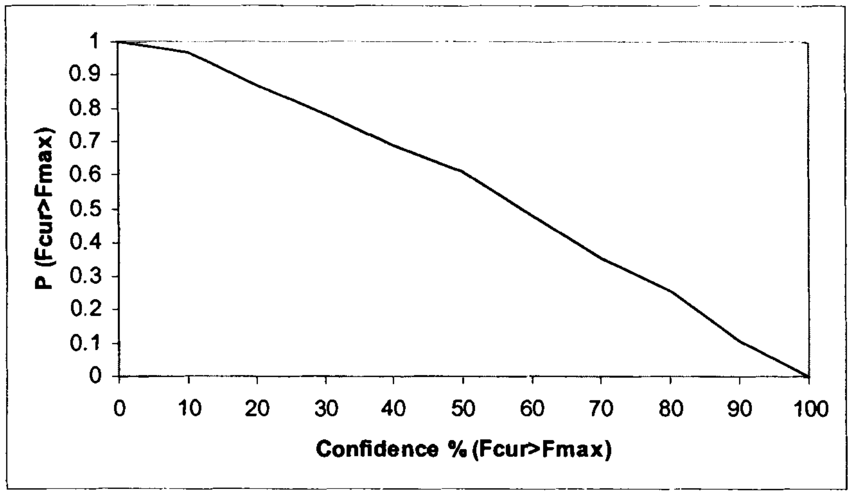


Figure 2.12. Simulation summary of the probabilistic estimate of F_{max} for the high-variation scenario (III). The layout is the same as in Figure 2.8.

Discussion

The 2001 stock assessment determined a probabilistic estimate of the current fishing mortality for the Maine sea urchin fishery, although no biological reference points had been derived. Therefore, without management targets or limits it is difficult to gauge the status of the sea urchin stock. In this study we investigated the use of $F_{0.1}$ and F_{max} as biological reference points in this fishery. Specifically, we were interested in how uncertainty in model parameters affects these reference points and the implications they have in assessing the stock status.

F_{max} corresponds to the fishing mortality that results in the highest YPR, and has been historically used as a management target (Ricker 1975). However, Mace (1994) demonstrated that F_{max} might not be ideally suited as a management target. Specifically, she found that F_{max} tended to exceed F_{msy} (fishing mortality that yields maximum sustainable yield), which calls into question its desirability as a management target. Therefore, since F_{max} may be too high for a target, it has been suggested for use as a management threshold (Mace 1994). However, F_{max}'s capacity as a management threshold is often limited because it is sensitive to variations in model parameters and it can be difficult to define or even indefinable (Rivard and Maguire 1993). F_{max} becomes indefinable when the instantaneous rate of growth exceeds the instantaneous rate of natural mortality (Mace 1994). In this situation, F_{max} is useless as a biological reference point, leaving the stock status assessment in ambiguity. F_{max} estimates can also be sensitive to variations in YPR analysis. In this study, the coefficient of variance was nearly three times greater for F_{max} than $F_{0.1}$ (Table 2.2.). We conclude that F_{max} should not be used as a management target biological reference point, and may not be a reliable management threshold biological reference point for the Maine sea urchin fishery.

$F_{0.1}$ appeared to be a more rational choice as a management target for several reasons. First, $F_{0.1}$ corresponds to a fishing mortality where increases in fishing intensity result in marginal increases in YPR. This means that $F_{0.1}$ is always less than F_{max}, and has comparatively lower YPR values. Therefore overexploitation of the stock is less likely to occur when using $F_{0.1}$, which is consistent with the FAO precautionary approach to fisheries management (Food and Agriculture Organization 1995). Second, $F_{0.1}$ is more robust to uncertainties in model parameters than F_{max} especially when YPR curves are poorly defined (Rivard and Maquire 1993). In this study, the coefficient of variations was much greater for F_{max} than $F_{0.1}$, although we did not observe significantly greater variability in F_{max} as parameter variation increased (Table 2.2.). This may be partly due to the non-linear estimation techniques used to estimate $F_{0.1}$ and F_{max}. The high variation scenario (III) had the highest variability in model parameters, which may have been more difficult to converge to a global minimum. This may explain why CV's for $F_{0.1}$ and F_{max} in scenario III are comparable to the low variation scenario (II). Third, $F_{0.1}$ is usually well defined and has been used extensively as a management target in other marine fisheries (Quinn and Deriso 1999). For these reasons we believe that $F_{0.1}$ is better suited to serve as the management target than F_{max} for the Maine sea urchin fishery.

Uncertainty in model parameters can have a large impact on the estimation of biological reference points (Helser et al. 2001; Chen and Wilson 2002). By incorporating uncertainties into both the stock assessment and biological reference points, we can investigate their combined effects on the stock status assessment. To achieve this we have used probability profiles, which present the current stock status inclusive of uncertainties in both the indicator and management reference points. Despite the variability in $F_{0.1}$ and F_{max} estimates caused by the uncertainty in model parameters, the probability profiles did not change substantially between scenarios (Figures 2.7., 2.8., 2.9., 2.10., 2.11. and 2.12.). Our interpretation of the stock status remains consistent with all levels of uncertainty. In all three scenarios, $P(F_{cur}>F_{0.1})$ remains at 1.00 for virtually all decision confidence levels. Therefore, we can conclude that the current fishing mortality is greater than the estimated $F_{0.1}$ for all tested variations. It is important to remember that biological reference points can be derived from a number of different models. Therefore the conclusions drawn from their use must be placed in context with their respective models.

This study is a preliminary investigation into the development of biological reference points for the Maine sea urchin fishery. We chose to use YPR analysis because it incorporates the effects of age of recruitment, natural mortality and growth rate and produces biological reference points that are widely used and accepted in marine fisheries management. Nonetheless, since other biological reference points represent different aspects of the fish stock structure and may result in different conclusions on the fishery, extensive studies into other models should be conducted before final selection and implementation of a biological reference point for monitoring and managing the Maine sea urchin fishery.

Chapter 3

ESTIMATING EXPLOITABLE STOCK BIOMASS USING A SPATIAL STATISTICS APPROACH

Chapter Abstract

The first formal stock assessment for the Maine green sea urchin fishery was conducted in 2001 using fisheries-dependent and biological data. The annual fishery–independent survey program was established in 2001 to provide spatially referenced fisheries-independent information. The objective of this study was to investigate the large-scale spatial patterns in sea urchin abundance to estimate the fishery's exploitable biomass. Triangulated irregular networks (TINs) were used to linearly interpolate urchin densities, characterizing large-scale patterns. The resulting density surfaces were modified to only include areas of the appropriate substrate type and depth zone and were used to calculate total biomass. Exploitable biomass was calculated as the total weight of legal-sized urchins that are at a density (10 urchins m^{-2}) high enough to be attractive to fishermen. Exploitable biomass was estimated at 5878 and 7101 metric tons for management zones 1 and 2, respectively. These estimates are almost identical to estimates calculated using a length-structured fisheries population dynamics model on fisheries-dependent data. We conclude that TINs are useful for stock biomass estimation in the green sea urchin fishery.

Introduction

The first formal stock assessment for the Maine green sea urchin fishery was conducted in 2001 (Chen and Hunter 2003). Fishery-dependent data and urchin life history parameters were used to assess the population dynamics of the Maine urchin stock. A length-based stock assessment model was used with a Bayesian approach to determine current stock biomass and exploitation rate. The study estimated that the current stock biomass was extremely low, about 10% of the virgin biomass.

Data sampling technique has a large impact on the quality of the stock assessment. The quality of fishery-dependent data is more questionable than fishery-independent data and its sole use in stock assessments may lead to large uncertainty or even bias (Hilborn and Walters 1992). Only fishery-dependent data were available for the first stock assessment (Chen and Hunter 2003), however in 2001 the DMR began an extensive fishery-independent survey program. This large, spatially referenced, scientific data set can be used in both traditional stock assessments and in those incorporating spatial analysis techniques.

Scientists have realized the importance of incorporating spatial variability into stock assessments and have adapted a number of spatial analysis techniques to explore spatial trends and to estimate or predict stock abundances. Moving averages, kernel estimation, spline methods, and tessellation are all spatial analysis techniques that can be used to estimate spatial patterns of population abundance (Ripley 1981; Bailey and Gatrell 1995).

Spatial statistics or spatial analyses are employed to model first and second order, or large and small-scale, spatial variability of a variable in order to estimate the value at unobserved locations (Bailey and Gatrell 1995; Petitgas 2001). Intrinsic second-order methods, along with kriging, have become the most popular geostatistical tools and are now commonly used to estimate exploited fish stock biomass (e.g., Simard et al 1992; Pelletier and Parma 1994; Maravelias 1996; Lembo et al. 1998; Maynou et al. 1998; Rivoirard et al. 2000; Petitgas 2001). First, the spatial distribution of the stock cannot be affected by the geometry of the region, *i.e.* the spatial distribution cannot differ near the borders of the zone (Petitgas 1993; Bailey and Gatrell 1995; Warren 1998; Rivoirard et al. 2000). Second, the process must exhibit some degree of second-order stationarity, or spatial dependence, which means that small-scale deviations in variables are similar in neighboring sites. If these assumptions are violated, we must use other spatial analysis techniques to estimate the spatial patterns.

Tessellation investigates first-order, or large-scale, spatial variability in a variable, meaning it estimates how the mean values vary over a study area (Peucker et al. 1976; Ripley 1981; Bailey and Gatrell 1995). The triangulated irregular network (TIN), or Delauney triangulation, is the simplest and most common tessellation technique for the creation of surfaces. The TIN surface consists of contiguous, non-overlapping triangles created by linear interpolation of the variable. TINs are most commonly used for visualization purposes but have been used to estimate stock biomasses (Simard 1992; Guan et al. 1999).

The objective of this study is to investigate the large-scale spatial trends in green sea urchin abundance using spatial analysis techniques in order to estimate biomass of the stock. This paper addresses how suitable TINs are for biomass estimation, specifically for the green sea urchin fishery. This study can provide the DMR with critical information on the sea urchin stock that can aid in the development of future management plans and help ensure a sustainable fishery.

Materials and Methods

Urchin density and size frequency information were obtained from the 2001 pilot study for the State's annual fishery independent survey. The Department of Marine Resources sampled 144 sites along the Maine coast using SCUBA. At each site, they randomly sampled 90 quadrats ($1m^2$) along a linear transect set perpendicular to shore. Sampling intensity was equally divided amongst three depth zones: 0-5m, 5-10m, and 10-15m. At each site, size frequency data were obtained by subsampling one quadrat per depth zone, in which test diameters were measured for all individuals in the quadrat. In the 15-40 m depth zone, an additional 148 sites were sampled using a video camera that recorded 10 quadrats ($0.5m^2$) at each site. Due to the low urchin densities, test diameters were measured for all recorded urchins. Mean urchin density values were calculated for each site (n=292), and by depth zone within each site (n=580).

Five test diameter categories were created to more accurately represent the wide range of individual urchin weights. The categories were based on the State's minimum and maximum size restrictions, allowing us to separately estimate the biomass of urchins that have not yet recruited to the fishery, urchins within the fishery, and urchins that have escaped the fishery. Urchin density values were scaled by the size frequency data, to generate urchin density values for each size category. Weight per urchin was calculated based on the mean length of the category using a length-weight relationship (Scheibling et al. 1999).

Representations of the large-scale trends in sea urchin density were created using the TIN method (ArcView 3.2a, 3D and Spatial Analyst Extensions). TIN surfaces were generated for 40 different scenarios, according to the size category, depth zone and management zone (Figure 3.1). The surface was then modified to only include areas of the appropriate depth and substrate type, using a customized C++ program. A map of surficial geology was used to identify areas of urchin habitats predominately including gravel or rock substrate (Kelley et al. 1999) (Figure 3.2). Digital gridded bathymetry data with 15 arc second resolution were used to create a plot of 5m isoline contours (Figure 3.3). This data source consists of digital bathymetry datasets from sources such as NOAA and the Naval Oceanographic Office (Roworth and Signell 2002). Modified urchin density plots were created for each scenario (Figure 3.4). The volume beneath the modified TIN surface was calculated, based on Riemann sums, and multiplied by the mean weight to determine total urchin biomass for each scenario. Fishable biomass is defined as the biomass of all legal sized urchins, and is simply the subset of the total biomass corresponding to legal sized urchins. Exploitable biomass was calculated by incorporating a threshold, or cut-off, density value for legal sized urchins, so volume was calculated only for areas with more than 10 urchins m^{-2}. This value was considered as the minimum density that could attract fishermen to fish in the urchin fishery. Estimation uncertainty was estimated using cross validation (n=60), which involves randomly removing sites and then modeling the process to calculate residuals. Urchin biomass values for depth zones and management zones were calculated based on the arithmetic mean to provide comparisons to the spatially derived estimates. Exploitation rates, or the ratio of commercial landings for the 2000-2001 fishing season to the exploitable biomass estimates, were calculated to facilitate comparison to the results generated from a traditional stock assessment (Chen and Hunter 2003).

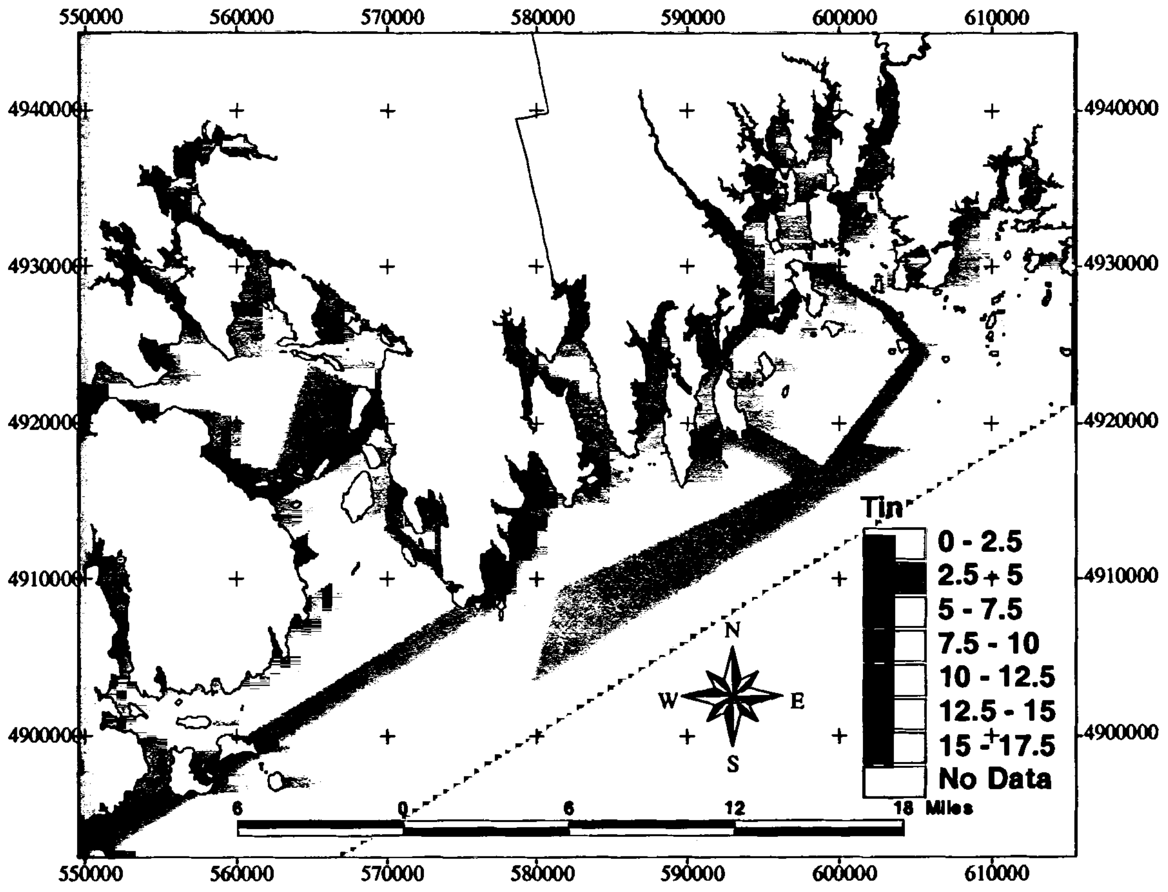


Figure 3.1. Plot of the urchin density surface for 50-65 mm urchins within the 0-5 m depth zone for the Mt. Desert Island area, in management zone 2, created using a triangulated irregular network (TIN).

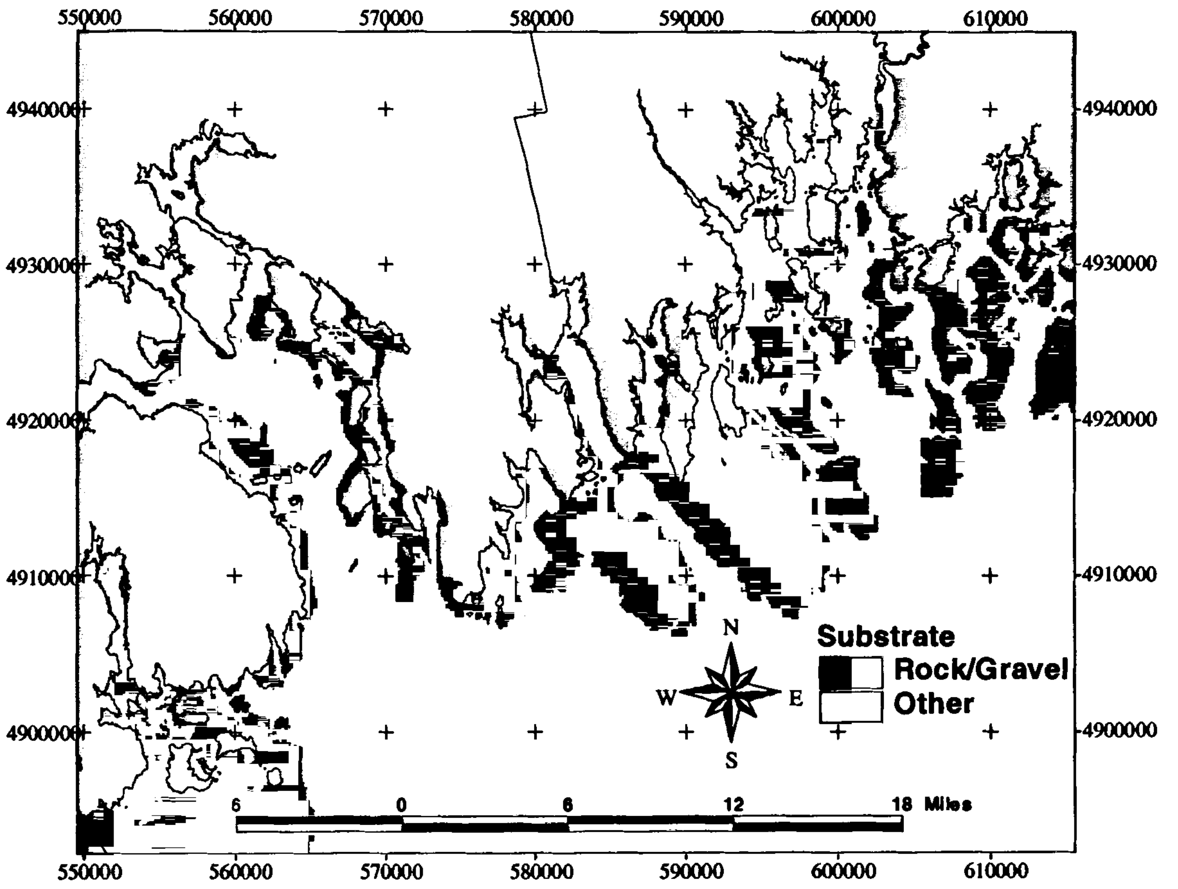


Figure 3.2. Plot of rock and gravel substrate within 40 m depth for the Mt. Desert Island area, in management zone 2.

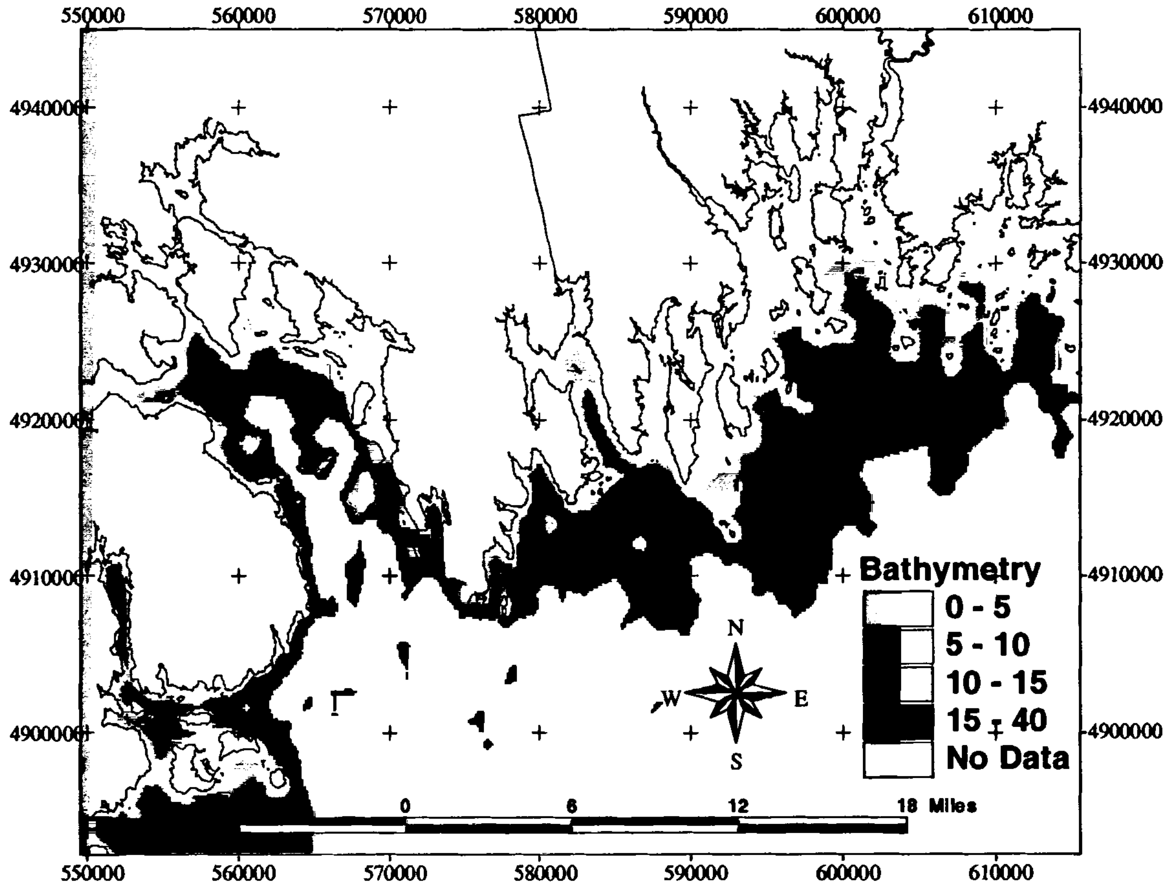


Figure 3.3. Plot of bathymetry (in meters) for the Mt. Desert Island area, in management zone 2, showing the 4 depth zones used in this study.

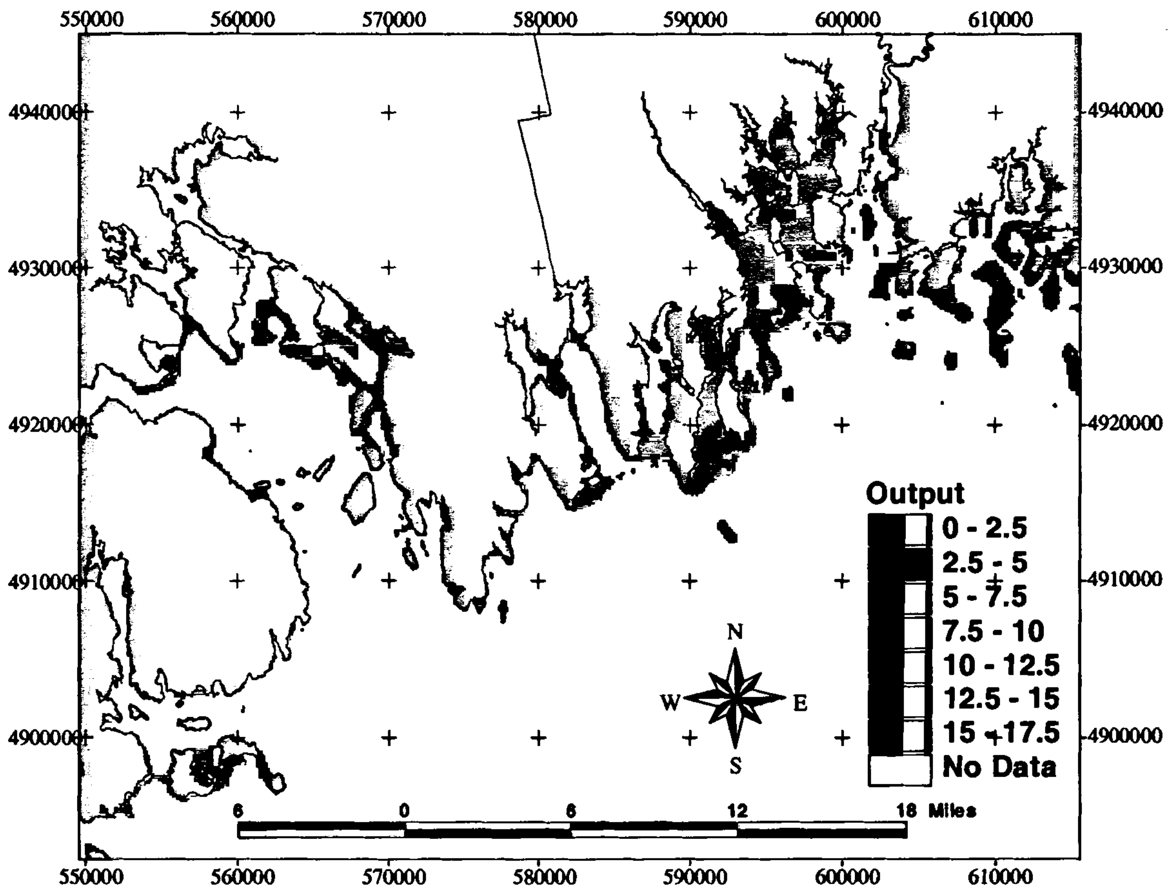


Figure 3.4. Plot of urchin densities for 50-65 mm urchins within the 0-5 m depth zone for the Mt. Desert Island area, in management zone 2. This plot was created by limiting the original triangulated irregular network (TIN) (Figure 3.1.) to areas of rock/gravel substrate (Figure 3.2.) within the 0-5 m depth zone (Figure 3.3)

Results

Sea urchin densities per m^2 and standard deviations varied by survey strata, being lowest in the western strata and highest in the eastern one (Table 3.1.). Average site densities had a mean number of 4.93 and were highly skewed to the right, with a skewness coefficient of 7.21 (Figure 3.5.). Mean density was similar for the three shallow depth zones with approximately 9.50 urchins m^{-2}, however the 15-40m zone was substantially less with 0.32 urchins m^{-2}. Urchin diameters varied from 8 mm to 114 mm with a mean at 35.90 mm (Figure 3.6.).

Table 3.1. Descriptive statistics for quadrat density counts (m^{-2}) by management zone and survey strata

Zone	Stratum	Density	St Dev	Density	Density	N
		Mean		Min	Max	
1	1	0.17	1.62	0	36	1706
1	2	2.57	10.63	0	130	1600
1	3	3.20	11.29	0	141	1580
2	4	4.20	14.13	0	180	1490
2	5	4.24	12.52	0	127	1580
2	6	10.06	17.59	0	147	1530
2	7	7.90	13.85	0	113	1498
2	8	13.50	20.38	0	113	1570
2	9	34.45	44.03	0	280	1540

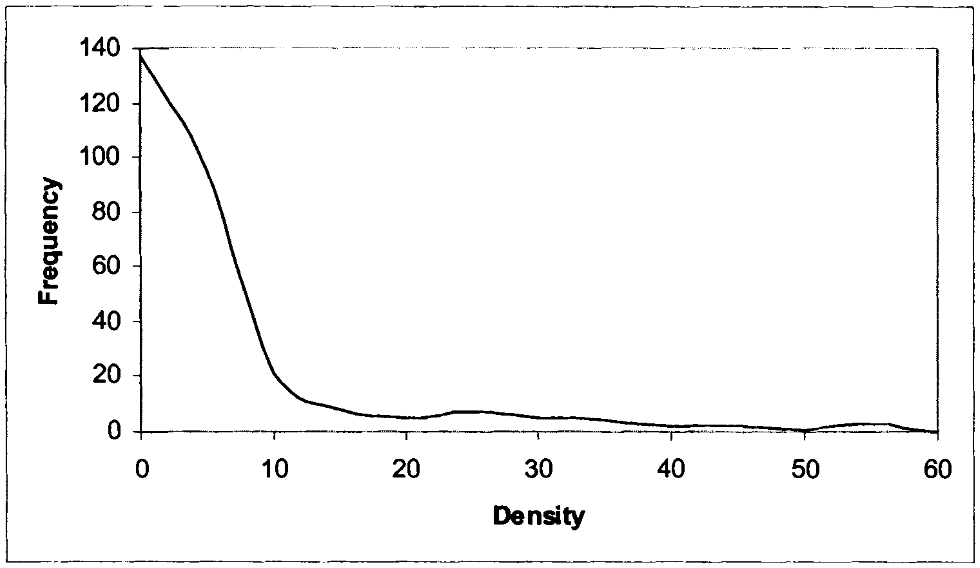


Figure 3.5. Histogram of urchin density values per m^2 for all sites, excluding one outlier with 174 urchins m^{-2}.

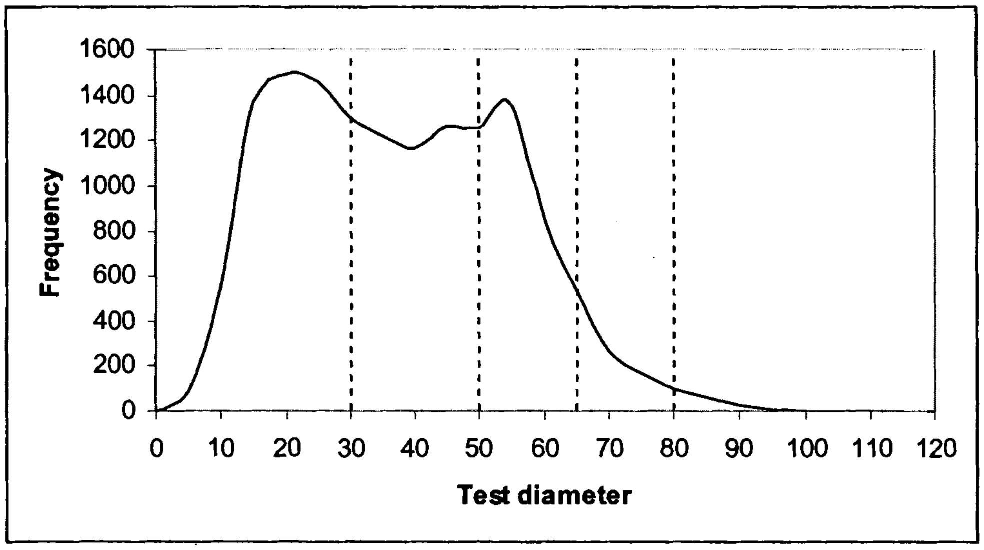


Figure 3.6. Histogram of urchin test diameters for all sampled quadrats. Dashed lines delineate the 5 size categories. Mean density is 35.90mm, with category mean densities of 18.5, 39.25, 55.35, 70.59, and 86.30mm.

Total sea urchin biomass was estimated at approximately 250,000 metric tons, in which over 80% was found in management zone 2 (Table 3.2.). Half of the total biomass was found in the 0-5m depth zone in management zone 2 (Figure 3.7.). The majority of the estimated total biomass was comprised of 50-80 mm and 39-64 mm urchins for management zones 1 and 2, respectively (Figure 3.8.). Fishable biomass was estimated at approximately 165,000 metric tons (Table 2.2.). The majority of the fishable biomass in each zone was within the 0-5m depth zone, with approximately 75% of the total fishable biomass within zone 2 (Figure 3.9.). There was no fishable biomass estimated for the 15-40 m depth zone for either management zone.

Patterns in exploitable biomass by depth differed from the patterns seen in total and fishable biomass (Figures 3.7., 3.9. and 3.10.). Over 95% of the fishable biomass in management zone 1 was found in the 0-5 m depth zone, which exceeded the corresponding estimate for management zone 2. However, the exploitable biomass estimate for management zone 2 was higher than zone 1 due to higher biomass estimates in the 5-10 m and 10-15 m depth zones.

The exploitable biomass estimates closely approximated the landings for the 2000-2001 fishing season (Table 2.2.). Exploitation rates calculated from the exploitable biomass estimates were 0.37 and 0.45 for management zones 1 and 2, respectively. Cross validation of sea urchin density surfaces estimated a mean residual of 0.50 (median=0, standard deviation=1.86, skewness=2.80, n=60).

Table 3.2. Summary of biomass estimates derived using the TIN method, the arithmetic mean, and a fishery population dynamics approach for 2001, in comparison to the 2000-2001 commercial landings.

	Zone 1	Zone 2	Total
TIN Method			
Total Biomass	45868	204304	250172
Fishable Biomass	39060	126725	165786
Exploitable Biomass	5878	7101	12979
Arithmetic Mean			
By Zone	74168	381809	455977
By Zone and depth	24409	242017	266426
Fish. Pop. Dynamics	6550	8452*	15002
2000-2001 Landings	2148	3213	5361

* 2000 value

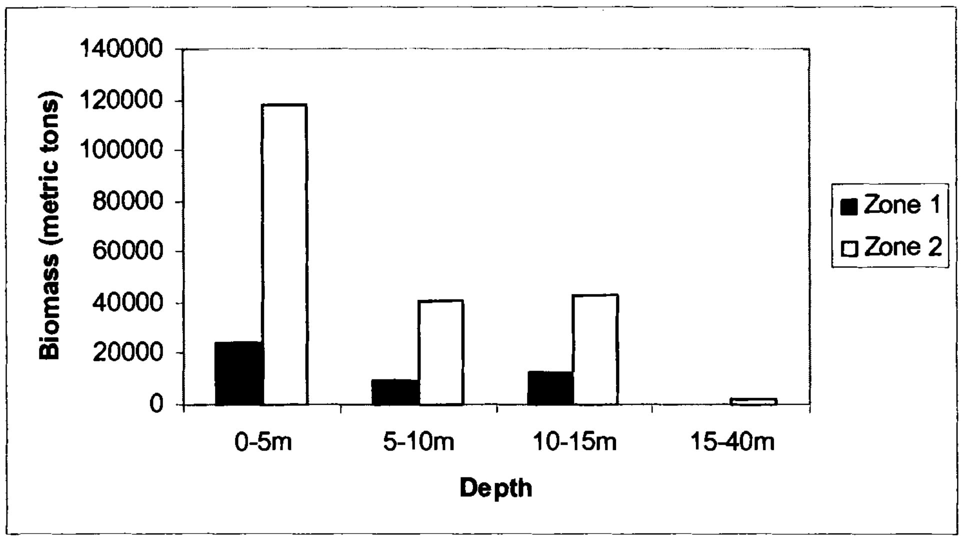


Figure 3.7. Total biomass estimates by depth zone according to management zone.

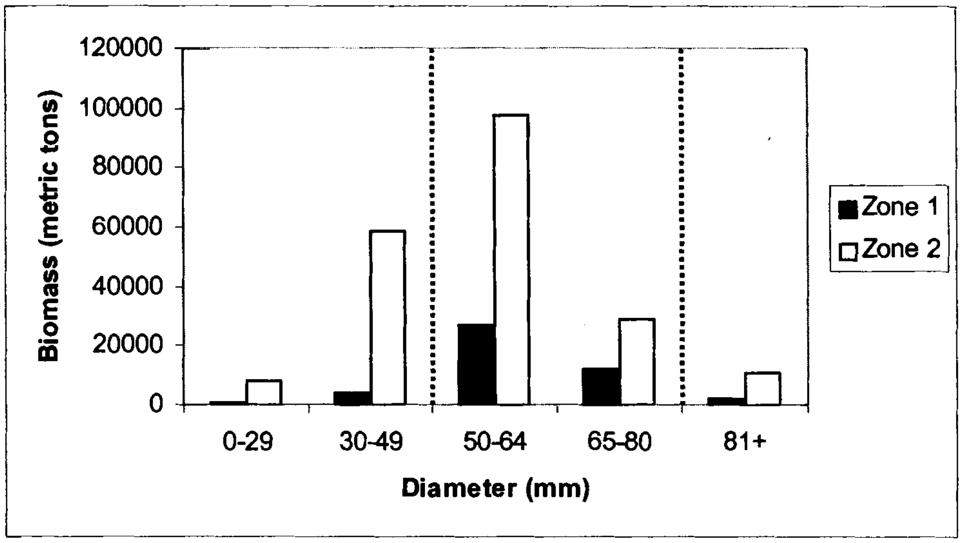


Figure 3.8. Total biomass estimates by sea urchin test diameter according to management zone. Urchins between 50 and 80 mm are legal sized urchins and constitute the fishable biomass.

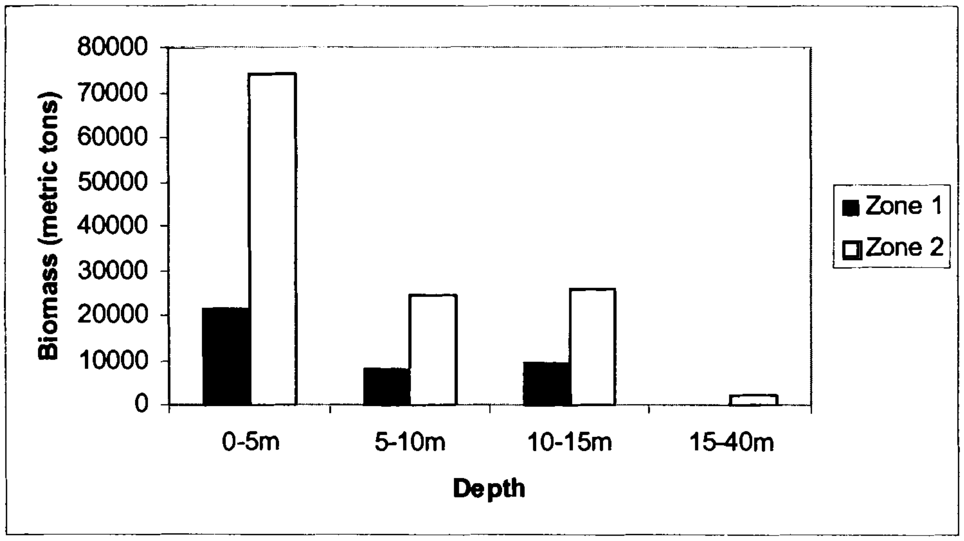


Figure 3.9. Fishable biomass estimates by depth zone according to management zone.

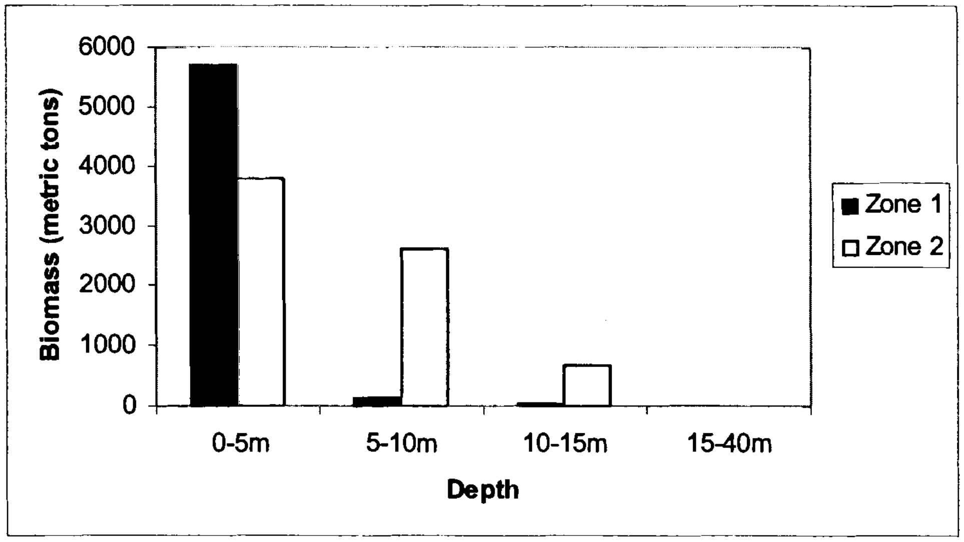


Figure 3.10. Exploitable biomass estimates (densities >10 urchins m^{-2}) by depth zone for the entire Maine coast.

Discussion

The objective of this study was to investigate the large-scale spatial patterns in sea urchin abundance to estimate the stock biomass. There are a number of spatial analysis techniques available that can generate this type of information. The most widely employed spatial analysis technique in fisheries is intrinsic geostatistics, which along with kriging can generate precise estimates of biomass based on the small-scale variations.

Intrinsic geostatistics are applicable to spatially auto-correlated processes based on two assumptions (Petitgas 1993; Bailey and Gatrell 1995; Warren 1998; Riviorard et al. 2000). First, the variable and the region's geometry are independent of one another. This cannot be assumed for sea urchin abundance data, which are not independent of the study area, but are dependent on the depth and substrate type. Second, there must be some degree of stationarity, either strict stationarity or stationarity of increments. The sea urchin data are highly skewed and spatially variable; stationarity does not exist so the variogram is an unreliable representation of the spatial continuity (Rossi et al. 1992). For example, an empirical variogram of site density means shows no spatial correlation (Figure 3.11.). There are ways to modify the data to make them more appropriate for variogram analysis, such as trend removal, lognormal transformation of data and stratification of the region based on variances (Rossi et al. 1992; Simard 1992; Maravelias 1996; Riviorard et al. 2000). However, these techniques are labor-intensive and provide marginal improvements in the variogram analysis for the sea urchin data. Since we cannot assume that the data is appropriate for intrinsic geostatistics, we used another spatial analysis technique, triangulated irregular networks, to characterize the large-scale spatial patterns in sea urchin abundance.

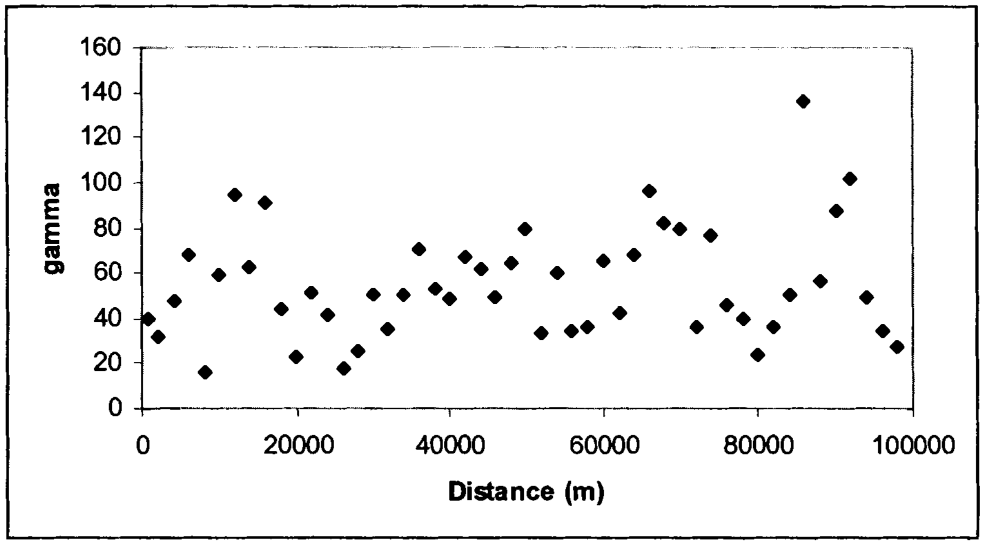


Figure 3.11. Sample empirical variogram representing the spatial covariance of average urchin density values per site, excluding one outlier with 174 urchins m^{-2}.

TINs are a simple spatial analysis technique for exploring the spatial variability of a process. They have received limited uses in fisheries stock assessment; however, because most stocks exhibit some degree of stationarity so intrinsic geostatistics provide more precise biomass estimates (Simard 1992; Guan et al. 1999; Bailey and Gatrell 1995). When sampling locations are relatively evenly spaced, as in the sea urchin abundances by depth zones, TINs are a good estimator of the large-scale spatial patterns for spatially uncorrelated processes (ESRI 1998; Guan et al. 1999). The technique requires no complex statistical decisions, making it an accessible tool for fisheries managers (Bailey and Gatrell 1995). However, this technique does not incorporate a variance structure into the estimation process, so uncertainty cannot be directly quantified. In this study, cross-validation showed that mean residuals of the modified density surfaces were greater than zero. This result suggests that there is a global bias in the TIN surfaces and that the biomass estimates were likely overestimated (Simard 1992). However, exploitable biomass estimates were probably less affected than total biomass estimates because the subdivision of abundance estimates by management zones, depth zones and size categories minimizes the spatial variability and the impact of outliers.

Exploitable biomass, not fishable biomass, was used as an estimation of the stock biomass (Table 2.2. and Figure 3.10.). This distinction was necessary because some areas incorporated in the estimation technique were not subjected to fishing pressure due to geographic isolation or low urchin densities. The exploitable biomass estimates accounted for this difference because a threshold density value of 10 urchins m^{-2} was incorporated into the analysis, upon recommendation by fishermen, ecologists and DMR fisheries biologists. Areas with densities below the threshold receive negligible commercial fishing pressure, and are not incorporated in the biomass estimations. Their impacts on the fishery, in particular in their contributions to the recruitment of the fishery, need to be assessed in future studies.

Biomass estimates generated using spatial analysis approaches were similar to ones generated using different assessment techniques. Total biomass estimated based on arithmetic means, stratified by zone and depth, was comparable to the TIN estimate for total biomass (Table 2.2.). Even more significant, the exploitable biomass estimates based on TINs were almost identical to ones calculated using traditional stock assessment techniques (Table 2.2.). This similarity is evident in the calculated exploitation rates, for which spatial analysis estimates (2001) were 0.37 and 0.45 for management zones 1 and 2, respectively, and the traditional techniques estimates were 0.38 (2001) and 0.57 (2000) for the management zones, respectively. These techniques are based on fundamentally different theories and the analyses used entirely different data sources; fishery-independent data for the spatial analysis assessment but fishery-dependent and biological data for the traditional fisheries stock assessment. Despite these differences, the biomass estimates and exploitation rates were almost identical. The implications of this lie in the status and management of the sea urchin fishery. Stock biomasses and exploitation rates can be used to gauge the stock status. The work on biological reference points (Chapter 2) suggests that current exploitation rates far exceed target reference points, signifying that the Maine sea urchin stock is over-fished. The confirmation of the two independent assessments not only validates the techniques, but also indicates that we are generating good biomass estimates and providing quality information to the stock managers.

Chapter 4

A SIMULATION FRAMEWORK FOR ESTIMATING OPTIMAL SAMPLING STRATEGIES

Chapter Abstract

Fishery-independent surveys are scientific studies that provide essential information for stock assessments and for developing appropriate management plans. Pilot studies are conducted prior to the start of a survey program, to gain information about the spatial distribution of the stock in order to optimize the survey. A pilot study for the annual fishery-independent survey program for the green sea urchin fishery was initialized in Maine in the summer of 2001. The pilot study was extensive, time-consuming and costly, and needed to be optimized to ensure its feasibility as a long-term scientific survey. The high degree of spatial variability in sea urchin abundance, however, prevented us from using standard optimization techniques, such as traditional statistics or even geostatistics. Kernel estimation and computer simulations were used to characterize the large-scale spatial density structure of the sea urchin population and investigate how different sampling strategies effected realizations of the density structure. Since realizations of the large-scale density structure are the vital components of the sea urchin stock assessment (Chapter 3), any changes in this structure would dramatically alter the outcome of the assessment. Therefore, we defined an optimal sampling strategy as a design that produces realizations of the large-scale spatial structure that are similar to the original population while using less sampling intensity than the original sampling strategy. Using the original survey strategy, a reduction of sampling intensity to 10 sites

per strata, or 90 total sites, was optimal because it corresponded to a large decrease in effort but only a marginal decrease in precision. However, a regular sampling strategy, with approximately 90 grids arranged along the coastline, provided the highest precision for the green sea urchin fishery when analyzed solely with spatial statistics. Considering that the sea urchin data will be analyzed by traditional and spatial statistics, we believe that the original stratified random sampling design reduced to 10 locations per strata is the most sensible optimization for the Maine green sea urchin fishery-independent survey program at this time.

Introduction

Fishery-independent surveys are scientific studies designed to provide biological and ecological information on a fish stock (Hilborn and Walters 1992; Jennings 2001). They can generate high quality data, with small variances and biases, which are representative of the entire targeted fish population. Stock assessments based on fishery-independent data have less uncertainty and bias than ones based on fishery-dependent data, which are generated from normal fishing activities. Therefore, fishery-independent surveys are essential for stock assessments and for developing appropriate management plans (Hilborn and Walters 1992). To establish an effective fishery-independent survey program, pilot studies should be conducted prior to the start of the survey program in order to gain information about the spatial distribution of the stock and to identify environmental variables that influence this distribution. The pilot study is then redesigned, or optimized, based on the information collected and on the future analysis plan (Andrew and Mapstone 1987; Kitsiou et al. 2001). According to Andrew and

Mapstone (1987), "Optimization of the design of sampling programmes is achieved by determining the most efficient allocation of resources-*i.e.*, minimizing decreases in precision and/or resolution imposed by cost or by logistical constraints."

A pilot study for an annual fishery-independent survey program was initialized in the summer of 2001 for the green sea urchin fishery in Maine. The pilot study was designed and implemented to provide detailed information on the population structure, spatial variability and biological/ecological characteristics of the sea urchin stock along the coast of Maine. The pilot study was extensive, time-consuming and costly, and could not be maintained for the annual survey. Therefore, the pilot study needs to be optimized to reduce the cost while maintaining high precision and accuracy of the annual survey.

Many statistical techniques have been developed to optimize sampling programs, including traditional experimental design, geostatistics and Monte Carlo computer simulation (Cochran 1977; Rivoirard 2000; Petitgas 2001). Traditional statistical methods are primarily based on random sampling and optimization usually involves stratification of the study area based on the spatial structure of the stock (Cochran 1977; Hilborn and Walters 1992). The study area is divided into smaller regions, or strata, using variables that influence the spatial structure of the stock, such as depth or habitat, in order to increase sampling precision and accuracy. Optimization with traditional statistics is limited, though, because these methods assume that the fish stock is distributed randomly over the study area or strata. Truly random distribution in a fished stock is rare, however, most stocks exhibit spatial patterns or dependence, also known as spatial heterogeneity. A different branch of statistics, known as spatial statistics, is specifically designed to investigate the spatial distribution of a stock and can be used for survey design optimization.

Spatial statistics or spatial analyses are employed to model first and second-order, or large and small-scale, spatial variability of a variable, such as fish abundance, in order to estimate the value at unobserved locations (Bailey and Gatrell 1995; Petitgas 2001). Intrinsic second-order methods have become the most popular geostatistical tools and the kriging variance, or mean square prediction error, can been used to compare survey designs for optimizing fishery surveys (Pelletier and Parma 1994; Rivoirard et al. 2000; van Groenigen 2000; Petitgas 2001). Two assumptions must be met in order to use intrinsic geostatistical methods. First, the spatial distribution of the stock cannot be affected by the geometry of the region, *i.e.* the spatial distribution cannot differ near the borders of the zone (Petitgas 1993; Bailey and Gatrell 1995; Warren 1998; Rivoirard et al 2000). Second, the process must exhibit some degree of second-order stationarity, or spatial dependence, which means that small-scale deviations in variables are similar in neighboring sites. In chapter 3, the suitability of the green sea urchin data for analysis with intrinsic geostatistics was addressed. The data did not satisfy the assumptions, especially for stationarity; the sea urchin data are too highly skewed and spatially variable. Since the assumptions are violated, we must use other spatial analysis techniques to characterize the spatial variability of the stock (Bailey and Gatrell 1995; Warren 1998; Petitgas 2001). .

Several spatial analysis techniques are available for investigating the large-scale variations in fish stock abundance (Bailey and Gatrell 1995). For example, in Chapter 3, triangulated irregular networks (TINs) were used to estimate exploitable biomass for the green sea urchin fishery. TINs are good estimators of large-scale spatial patterns but require relatively evenly spaced sampling locations; its performance decreases when sampling locations become clustered (ESRI 1998; Guan et al 1999). Kernel estimation is an advanced form of weighted spatial moving averages that can be used with any type of sampling strategy: random, clustered or grids (Bailey and Gatrell 1995). It does not require any major assumptions nor does it require complex statistical decisions or modeling. Therefore, kernel estimation is used to estimate the large-scale patterns in sea urchin stock abundance, but since it does not incorporate a variance structure, it cannot be directly used for sample design optimization.

Kernel estimation paired with computer simulations may provide the framework necessary for optimizing survey programs. Computer simulation approaches have been increasingly used in fisheries due to their ability to incorporate different sources of variations, especially spatial and temporal heterogeneity (e.g. Hilborn and Walters 1987; Horppila and Peltonen 1992; Andrew and Chen 1997). A simulation approach allows researchers to investigate how uncertainty in the spatial structure of a fished stock can affect survey programs and stock assessments. Sampling programs based on random sampling theory can have countless realizations, and the precision of one realization may not represent the precision of the sampling program. Simulations allow us to produce multiple realizations and estimate the mean precision of a sampling strategy.

The objective of this project is to develop a framework that incorporates spatial statistics and computer simulations to identify an optimum sampling strategy. An optimal sampling strategy should provide the most accurate and precise information on a stock, as possible. Since we are using spatial statistics, we are most interested in the large-scale spatial structure of sea urchin density. The combination of kernel estimation and computer simulation allows us to estimate the large-scale spatial density structure and determine how different sampling strategies effect realizations of this structure. Since these realizations are the vital components of the sea urchin stock assessment (Chapter 3), any changes in these structures would dramatically alter the outcome of the assessment. Therefore, we define an optimal sampling strategy as a design that produces realizations of the large-scale spatial structure that are similar to the original population while using less sampling intensity than the original sampling strategy.

Materials and Methods

Urchin density and size frequency information were obtained from the 2001 pilot study for the State's annual fishery independent survey. The Department of Marine Resources employed a stratified random sampling design, where 16 sites were sampled in each of 9 strata along the Maine coast exclusively in potential urchin habitat (rock or gravel substrate) (Figure 4.1.). To minimize the sample variances within the strata, the width of each stratum was inversely proportional to the commercial landings in the region. At each site, 90 quadrats ($1m^2$) were randomly sampled along a linear transect set perpendicular to shore using SCUBA. All urchins within the quadrat were counted and test diameter was measured. Sampling intensity was equally divided over three depth zones: 0-5m, 5-10m, and 10-15m. Mean site densities were calculated, as were mean site densities by depth zones to allow each depth stratum to be analyzed separately.

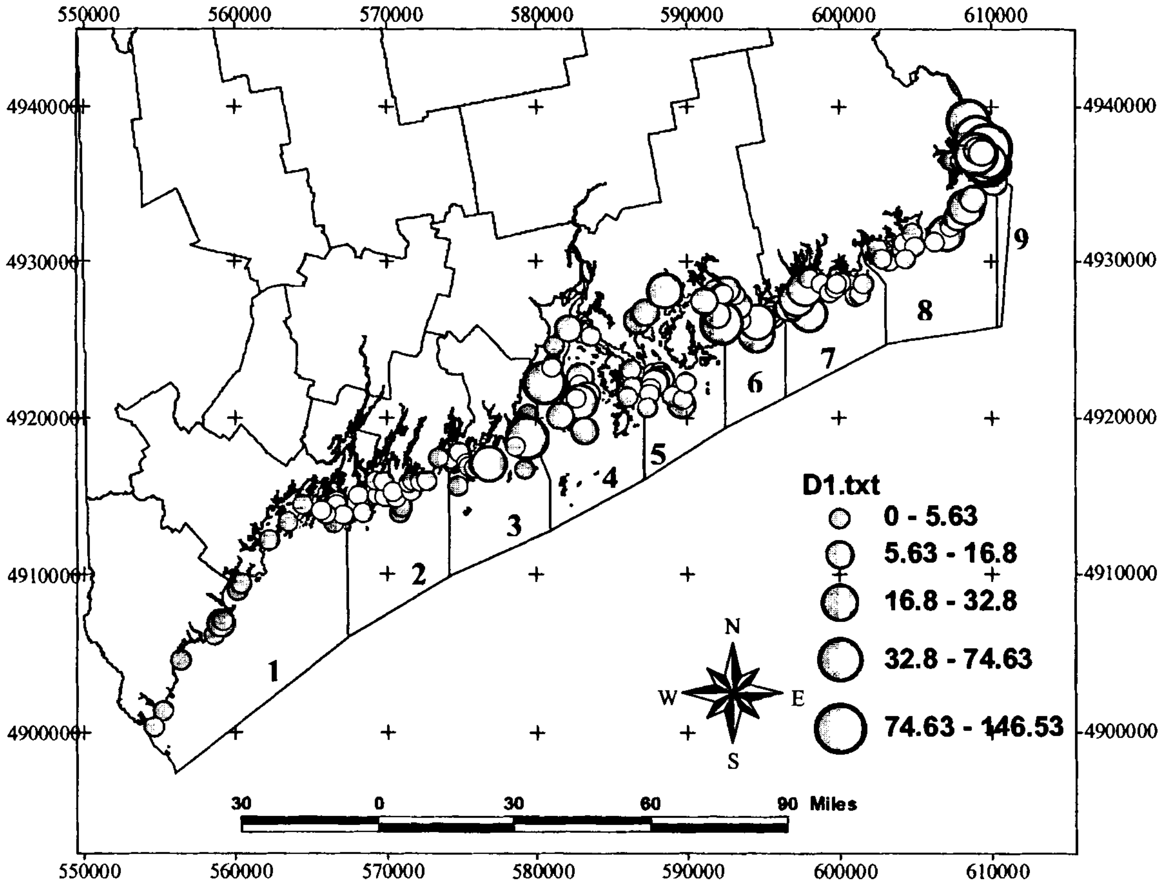


Figure 4.1. Mean urchin densities (urchins m^{-2}) for the 0-5 m depth zone from the 2001 pilot study. Strata from the original stratified random strategy are labeled.

A simulation framework was developed to test the ability of different sampling programs to recreate the large-scale spatial structure of the sea urchin population (Figure 4.2.). Mean sea urchin densities by depth zone, as well as bathymetry and suitable urchin substrate data, were the initial inputs for the framework. Kernel estimation was used to estimate the large-scale variations in the green sea urchin stock by depth zone (Bailey and Gatrell 1995). The kernel estimate for mean urchin density at a location is calculated as

$$\hat{\mu}_\tau = \frac{\sum_{i=1}^{n} k(\frac{s-s_i}{\tau})y_i}{\sum_{i=1}^{n} k(\frac{s-s_i}{\tau})}, \tag{1}$$

where $\hat{\mu}_\tau$ is the mean urchin density; k is the kernel, or bivariate probability function; **s** is the location (x,y) where the urchin density is being estimated; $\mathbf{s}_i$ are the locations where the urchin densities were sampled; τ is the bandwidth, or the radius of the moving window; and y_i is the urchin density. The study area was converted into an ASCII raster image (1500 x 1178 pixels, pixel=236.93 m) and weighted averages were computed for every pixel based on a quartic kernel. A bandwidth, in pixels, was selected to minimize error and ensure adequate coverage and smoothness. The kernel estimation technique produced plots of smoothed urchin densities by depth zone. These plots were modified to only include areas of the rock/gravel substrate, in effect producing spatial representations of the population density structure (Figure 4.3).

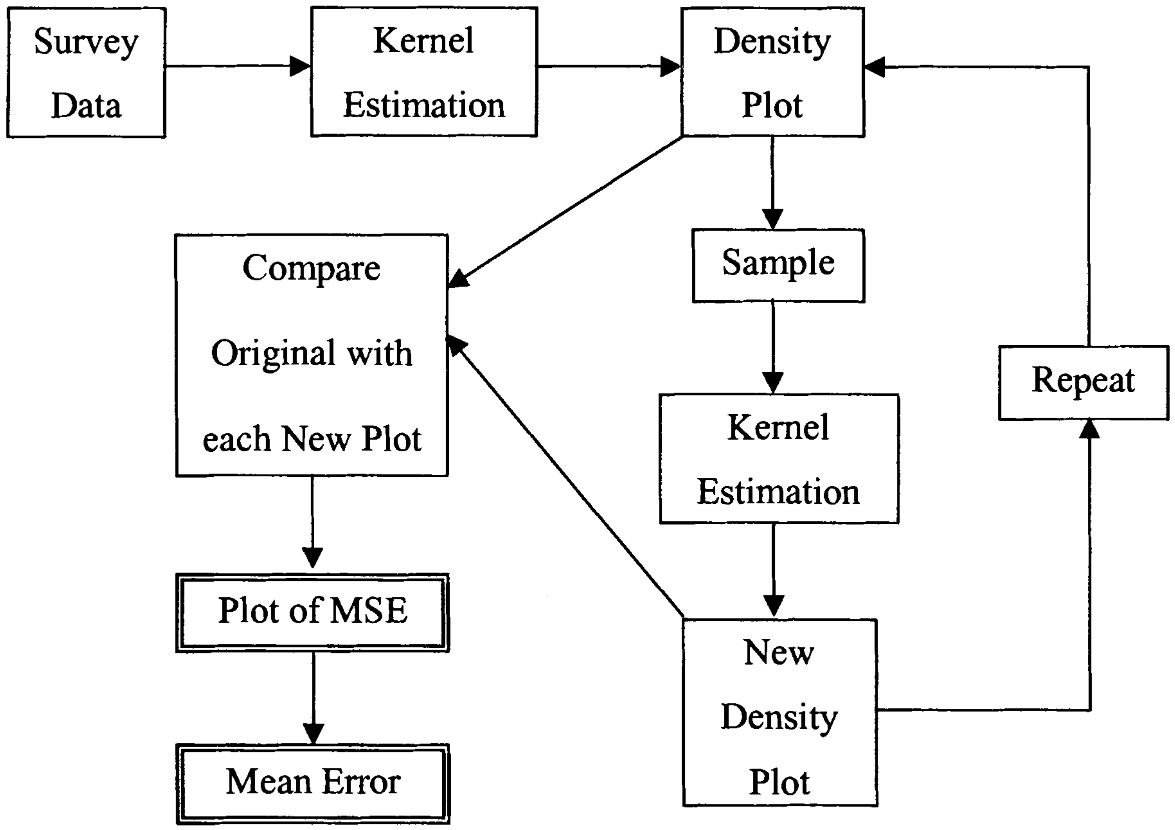


Figure 4.2. Flowchart of simulation approach to estimate the variance associated with a sampling strategy for the Maine sea urchin fishery.

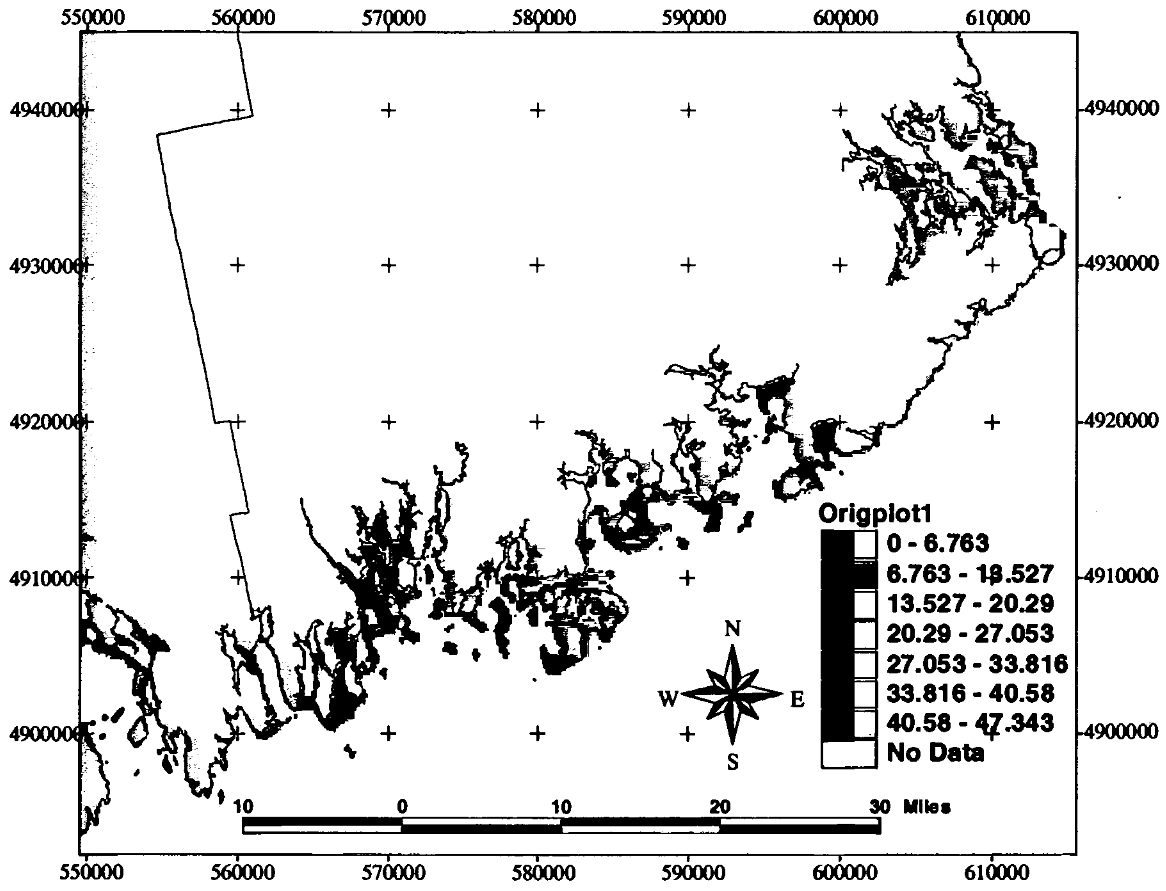


Figure 4.3. Original density plot characterizing the large-scale spatial variations in stock for density (urchins m^{-2}) depth zone 1 (0-5m) in areas west of Mt. Desert Island.

These original density plots were used to test different sampling strategies and gauge their relative performance. The sampling strategies varied based on the number of sites and number and size of strata, allowing us to test the following survey designs: random, stratified random with equal strata width, and stratified random with strata based on the original survey design (Table 4.1.). These sampling designs were chosen because they were feasible for the program and are routinely used in fishery surveys. Resampling was conducted randomly within the potential urchin habitat in the appropriate depth zone, producing sets of urchin densities by location. New urchin density plots were created from these observations using the kernel estimation technique. The number of simulations was limited to 50 due to restraints placed on computing power imposed by the large size of the files.

The performance of a sampling strategy was evaluated using mean squared error (MSE). The MSE has been used to determine optimal sampling strategies for fisheries and is calculated as

$$MSE(Q) = \frac{\sum_{N=1}^{N}(Q_S - Q_O)^2}{N}, \tag{2}$$

where Q_O is the stock density value from the original density plot, Q_S is the stock density value from the sampled plot, and N is the number of simulations (Cochran 1977; Guan et al. 1999). MSE was calculated for each pixel in the urchin density plots (n=1,767,000), creating a plot of MSE for each sampling strategy. A mean MSE value was calculated for each plot from the pixel MSE values to facilitate selection of an optimal sampling strategy. An optimal sampling strategy is a design that minimizes mean MSE while using less sampling intensity than the original pilot study.

Table 4.1. Summary of the sampling strategies evaluated in this study.

Sampling strategy	Number of strata	Sites per strata
Original Stratified Random Strata width dependent on landings	9	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
Random	1	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150
Stratified Random Equal strata width	2	6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72
	3	4, 8, 12, 16, 20, 24, 28, 32, 36. 40, 44, 48
	4	3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36
	5	3, 6, 9, 12, 15, 18, 21, 24, 27, 30
	6	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24
	7	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22
	8	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18

For the stratified random strategies with equal strata width, strata are defined as equal subdivisions of the coast along an east-west axis.

Table 4.1. Contd

Sampling strategy	Number of strata	Sites per strata
Stratified Random	9	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
Equal strata width		12, 13, 14, 15, 16
	10	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
		12, 13, 14, 15
	11	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
		12, 13, 14
	12	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
		12
	15	6
	18	5
	30	3
	45	2
	90	1

Results

The first kernel estimation step produced the original density plots, which characterizes the large-scale spatial variations in the sea urchin stock (Figure 4.3.). After implementing a sampling strategy, the second kernel estimation step created the sampled density plots (Figure 4.4.). Finally, plots of MSE were created for each scenario by calculating MSE per pixel (Figure 4.5.).

The stratified random strategy from the pilot study was tested using the 3 depth zone datasets and an average site dataset. MSE values for depth zone 1, 0-5 m, were considerably higher than the other datasets (Figure 4.6.). This result suggested that depth zone 1 had the highest spatial variability, so recreations of the large-scale variations in urchin density were the least precise. This dataset was used in all subsequent analyses because it is the most variable urchin density structure; it represents a worst-case scenario. A reduction of sampling intensity to 10 sites per strata, or 90 total sites, corresponded to a large decrease in effort but only a marginal decrease in precision (Figure 4.6.). MSE at a sampling intensity of 90 sites was used as a reference point for comparison between sampling strategies.

None of the tested sampling strategies had consistently lower MSE values than the original pilot study design, over all sampling intensities (Figure 4.7.). At low sampling intensities (less than 27 sites) random sampling had the lowest MSE. At greater than 27 sites, the original survey had the lowest MSE values over the majority of sampling locations. However, when sampling intensity was set at 90 sites, MSE values for the stratified random strategies with equal strata width dropped below the original survey design at higher levels of stratification.

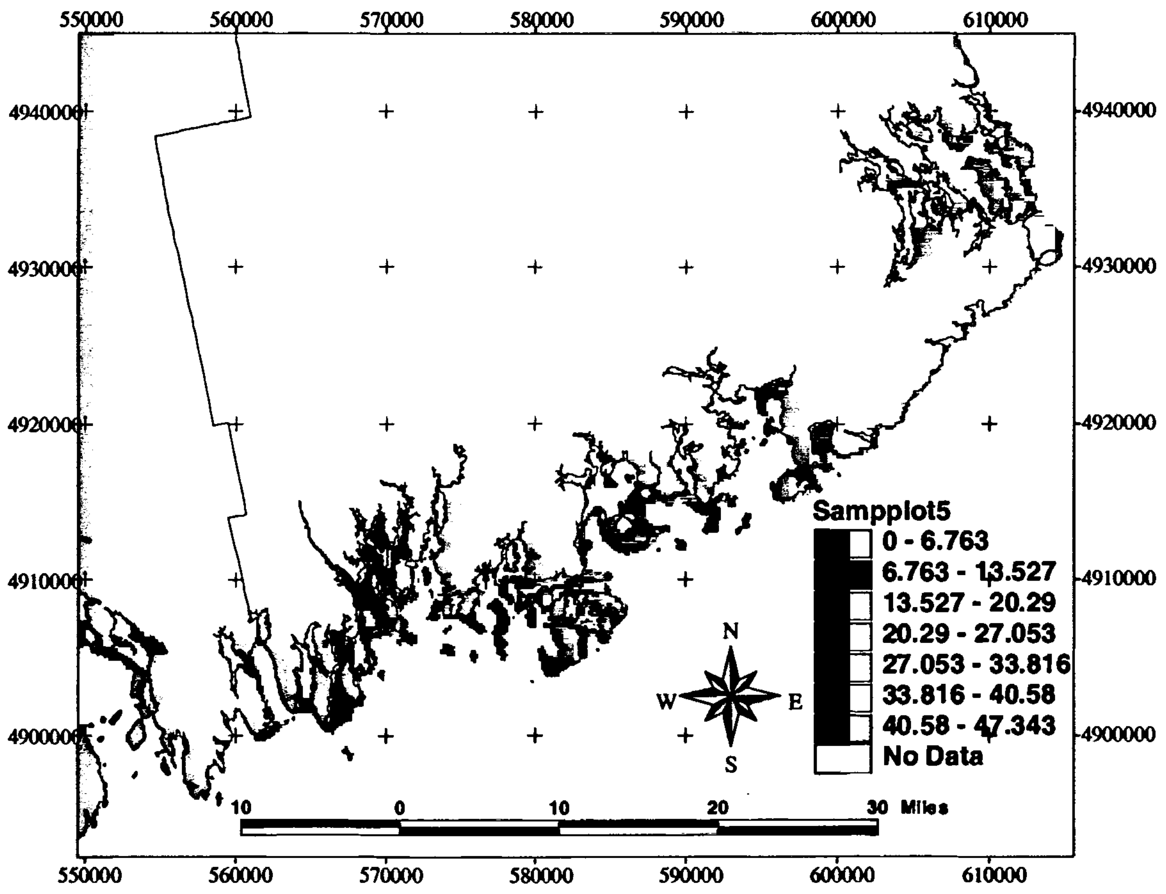


Figure 4.4. One simulation of a sample density plot (urchins m^{-2}) created by sampling the original density plot with the original stratified random design using 10 sites per strata in areas west of Mt. Desert Island.

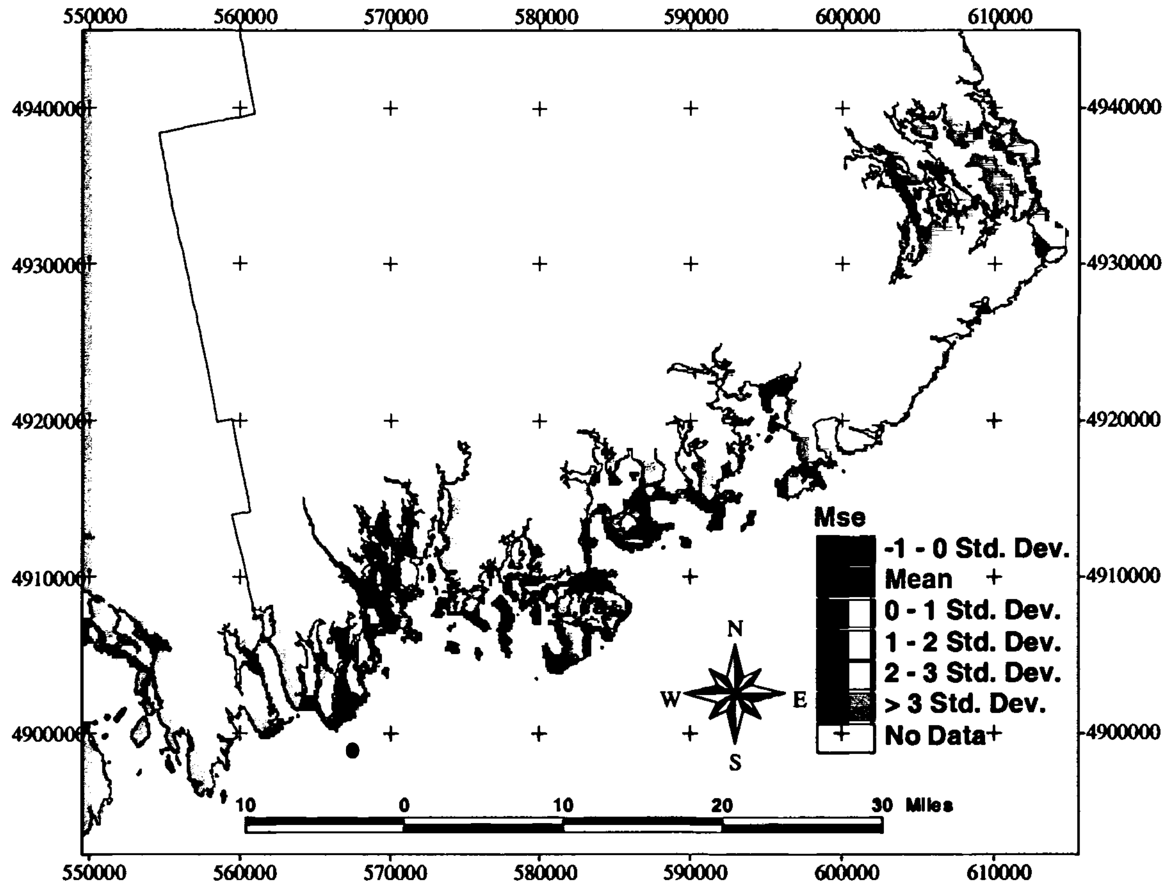


Figure 4.5. Plot of MSE for the original stratified random design using 10 sites per strata in areas west of Mt. Desert Island. Mean MSE is 2.90.

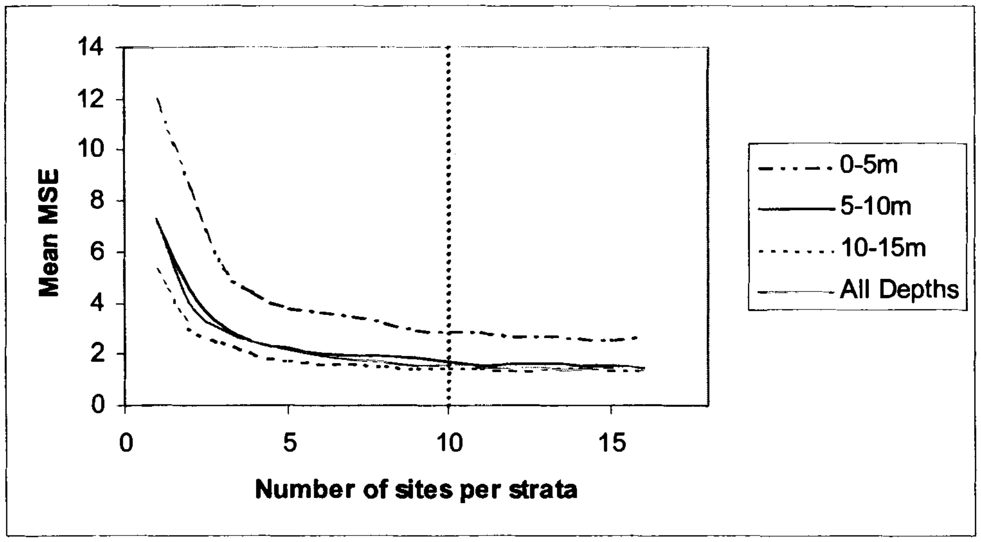


Figure 4.6. Mean squared error (MSE) as a function of the number of sites sampled per strata, using the original survey design by depth zone. The dashed line represents 90 sampling locations, 10 sites in each of 9 strata, which was used for comparisons amongst different sampling strategies.

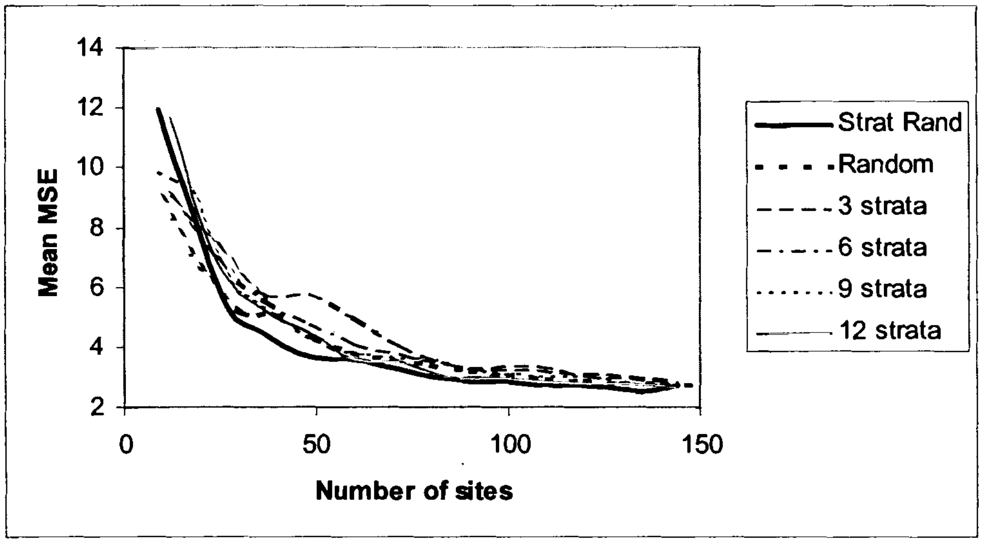


Figure 4.7. Mean squared error (MSE) as a function of the number of sites for the original stratified random sampling strategy (Strat Rand), random sampling, and stratified random sampling with equal strata width (3-12 strata) for depth zone 1 using the simulation framework approach.

At 90 sites, sampling strategies with greater than 9 equal sized strata had lower MSEs than the original survey design (Figure 4.8.). MSE values decreased with increasing stratification, reaching a minimum at 45 strata with 2 sampling locations. The original survey strategy performed better, with 90 sites, than random sampling (1 strata) and all stratified random strategies with less than 9 equal sized strata.

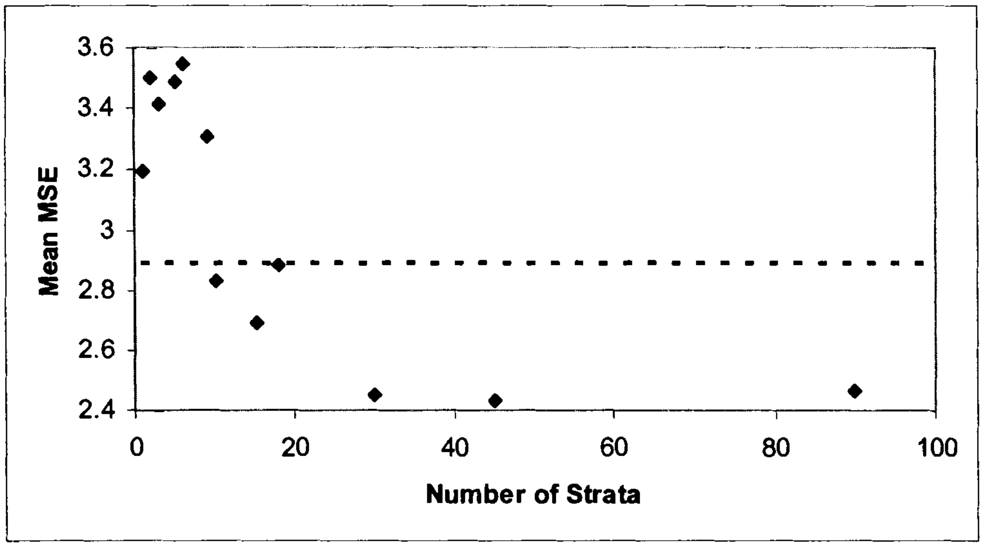


Figure 4.8. Mean squared error (MSE) for stratified random sampling strategies with equal strata width using 90 samples. The dashed line represents the MSE for the original sampling strategy with 90 sites, 10 in each of the 9 unequally sized strata.

Discussion

In optimization studies, we assume that the population was oversampled so the data collected is representative of the entire population. We believe that this assumption is valid for the 2001 pilot study for the green sea urchin fishery-independent survey; therefore we can legitimately optimize the survey. We defined an optimal sampling strategy as a design that produces realizations of the large-scale spatial structure that are similar to the original population while using less sampling intensity than the original sampling strategy. Within the original survey design, MSE quickly decreased and leveled off as sampling intensity increased (Figure 4.6.). We chose 90 sites as a reference point for this sampling strategy because it corresponded to a large decrease in sampling effort, a marginal increase in MSE and was buffered from the high MSE values at lower sampling intensities. When comparing amongst other sampling designs, however, the original sampling strategy did not have the lowest MSE.

In our study, the stratified random strategy with equal strata width had comparable or higher precisions than the original stratified random strategy. In particular, sampling strategies with more than 9 equal sized strata had considerably lower MSE values than the original sampling strategy (Figure 4.8.). Interestingly, MSE decreased further with added stratification. The high levels of stratification most likely caused this increase in precision. As the number of strata increased, and correspondingly the number of sampling locations per strata decreased, the sampling strategy more closely resembled a regular, or grid, sampling strategy. Grids have long been considered ideal sampling strategies for analysis with spatial statistics (Haining 1990, Rivoirard et al. 2000; Petitgas 2001). In fact, as long as the spatial process is not periodic, grids are the preferred option (Haining 1990; Simard et al. 1992). Accordingly, a regular sampling strategy, with 90 grids arranged along the coastline, would provide the highest precision for the green sea urchin fishery when analyzed with spatial statistics.

Currently the green sea urchin fishery is not analyzed solely with spatial statistics, though. Fishable biomass was estimated with spatial statistics (Chapter 3), while stock assessments (Chen and Hunter 2003) and investigations into biological reference points (Chapter 1) have been conducted using fisheries population dynamics and computer simulation techniques. Therefore, the optimal sampling strategy not only needs to satisfy the original criteria, *i.e.* minimizing decreases in precision while reducing sampling intensity, but, additionally, must be suitable to the future analysis plans (Andrew and Mapstone 1987). A regular sampling strategy may be the preferred design for analyzing the sea urchin stock with spatial statistics but it is not preferred for traditional statistics. When used with traditional statistics, regular sampling strategies can yield greater precision, but estimates are usually biased and sample variance cannot be directly estimated from the samples (Cochran 1977: Hilborn and Walters 1992). Conversely, stratified random sampling strategies are appropriate for both traditional statistics and spatial statistics. In traditional statistics, stratified random strategies have greater precision than random designs if the variance of a variable per strata is less than the overall variance (Hilborn and Walters 1992). In spatial statistics, stratified random strategies can have lower variances than random and grid designs, especially if there is a spatial trend (Haining 1990). So, a careful designed stratified random strategy, where strata size reduces sampling variance, would be more flexible for analysis than a regular sampling strategy.

An optimal sampling strategy must balance many factors, ranging from logistics and cost, to precision and analysis techniques. We believe that the original stratified random sampling strategy with reduced sites per strata is the best compromise and a sensible optimization for the Maine green sea urchin fishery-independent survey program at this time. Further simulation work on optimization should continue in order to investigate different sampling designs using more simulations.

Chapter 5
CONCLUSIONS

The objective of this thesis was to utilize fishery-dependent data and a fishery-independent survey to generate critical information for the management and monitoring of the Maine green sea urchin fishery. In particular there were three main areas of interest: (1) investigation of biological reference points, (2) spatial analysis and biomass estimation, and (3) development of a simulation framework approach to determine an optimal sampling strategy for the fishery-independent survey program.

These topics are intrinsically linked with one another. In Chapter 2, we calculated a target fishing mortality rate for the Maine sea urchin fishery. We used a Monte Carlo simulation approach on fishery-dependent data to estimate uncertainties of $F_{0.1}$ and F_{max}. This work suggested that $F_{0.1}$ is more suitable as a management target than F_{max} because it is precautionary, more robust to estimation uncertainty and usually well defined. We concluded that the current fishing mortality was greater than the estimated $F_{0.1}$ for all tested variations; in other words, the stock was overfished. In Chapter 3, we estimated the exploitable biomass and the current exploitation rate using the fishery-independent survey and spatial analysis techniques. These estimates were almost identical to estimates calculated using a length-structured fisheries population dynamics model on fisheries-dependent data (Chen and Hunter 2003). These techniques are based on fundamentally different theories and the analyses used entirely different data sources; fishery-independent data for the spatial analysis assessment but fishery-dependent and biological data for the traditional fisheries stock assessment. Despite these differences, the biomass estimates and exploitation rates were almost identical. The fact that we have two equivalent, independent assessments of the current stock indicates that we are generating realistic biomass estimates and providing quality information to the stock managers. This convergence is also significant because it strengthens our conclusions that the current stock is overfished. However, this status depends heavily on the biological reference point used in the study. Different biological reference points indicate different aspects of the fish stock structure and may result in different conclusions on the fishery status. Therefore, we suggest that further investigations into other reference points be conducted before final selection and implementation of a management target for the Maine sea urchin fishery.

Finally, in Chapter 4, we recognized the importance of including fishery-independent data in stock assessments, because they decrease uncertainty and bias in the estimates. An annual fishery-independent survey must be carefully designed to ensure statistical integrity, but it must also be cost and time effective to ensure its feasibility and longevity. An optimal sampling strategy must also address what types of analysis techniques will be used on the data. Considering that the data will be analyzed by both traditional and spatial statistics, we concluded that the original stratified random sampling design reduced to 10 locations per strata was the most sensible optimization for the Maine green sea urchin fishery-independent survey program at this time.

The research presented in this thesis will hopefully provide the DMR with essential information on the sea urchin stock, suggest new analysis techniques, and recommend a cost and time effective plan for collecting quality long-term fishery-independent data.

REFERENCES

Andrew NL, Chen Y. 1997. Optimal sampling for estimating the size structure and mean size of abalone caught in a New South Wales fishery. Fishery Bulletin 95:403-413.

Andrew NL, Mapstone BD. 1987. Sampling and the description of spatial pattern in marine ecology. Barnes M, editor. Aberdeen, U.K.: Aberdeen University Press. 39-90 p.

Bailey T, Gatrell A. 1995. Interactive Spatial Data Analysis. Essex, England: Pearson Education. 413 p.

Chen Y. 1996. A Monte Carlo study on the impacts of the size of subsample catch on estimation of fish stock parameters. Fisheries Research 26: 207-225.

Chen Y, Hunter M. 2003. Assessing the green sea urchin (Strongylocentrotus droebachiensis) stock in Maine, USA. Fisheries Research 60:527-537

Chen Y, Wilson C. 2002. Estimating uncertainty associated with biological reference point F0.1 for the American lobster (*Homarus americanus*) fishery in the Gulf of Maine and some possible management implications. Canadian Journal of Fisheries and Aquatic Sciences 59:1394-1403.

Cochran, WG. 1977. Sampling Techniques. New York: John Wiley. 428 p.

Deriso RB. 1987. Optimal F0.1 criteria and their relationship to maximum sustainable yield. Canadian Journal of Fisheries and Aquatic Sciences 44(Suppl. 2):339-348.

Environmental Systems Research Institute, Inc. (ESRI). 1998. ARC/INFO Version 7.2.1. Redlands, CA.

Fogarty MJ, Mayo RK, O'Brien L, Serchuk FM, Rosenberg AA. 1996. Assessing uncertainty in exploited marine populations. Reliability Engineering and System Safety. 54:183-195.

Food and Agriculture Organization (FAO). 1995. Precautionary approach to fisheries. Part I: Guideline on the precautionary approach to capture fisheries and species introduction. FAO Technical Paper 350:52.

Francis RICC, Shotton R. 1997. "Risk" in fisheries management: a review. Canadian Journal of Fisheries and Aquatic Sciences 54:1699-1715.

Guan W, Chamberlain RH, Sabol BM, Doering PH. 1999. Mapping Submerged Aquatic Vegetation with GIS in the Caloosahatchee Estuary: Evaluation of Different Interpolation Methods. Marine Geodesy. 22(2):69-92.

Haining R. 1990. Spatial Data Analysis in the Social and Environmental Sciences. Cambridge: Cambridge University Press.

Helser TE, Sharov T, Kahn DM. 2001. A stochastic decision-based approach to assessing the Delaware Bay blue crab (*Callinectes sapidus*) stock. In: Berkson JM, Kline LL, Orth DJ, editors. Incorporating uncertainty into fishery models. Bethesda, MD: American Fisheries Society Publications.

Hilborn R, Pikitch EK, Francis RC. 1993. Current trends in including risk and uncertainty in stock assessment and harvest decisions. Canadian Journal of Fisheries and Aquatic Sciences 50:874-880.

Hilborn R, Walters C. 1987. A general model for simulation of stock and fleet dynamics in spatially heterogeneous fisheries. Canadian Journal of Fisheries and Aquatic Sciences 44:1366-1369.

Hilborn R, Walters CJ. 1992. Quantitative fisheries stock assessment: Choice, dynamics and uncertainty. New York: Chapman and Hall. 570 p.

Horppila J, Peltonen H. 1992. Optimizing sampling from trawl catches: contemporaneous multistage sampling for age and length structures. Canadian Journal of Fisheries and Aquatic Sciences 49:1555-1559.

Jennings S, Kaiser M, Reynolds JD. 2001. Marine Fisheries Ecology. Oxford: Blackwell Science. 417p.

Kelley JT, Barnhardt WA, Belknap DF, Dickson SM, Kelley AR. 1999. The seafloor revealed. Maine Geological Survey. 55 p.

Kitisiou D, Tsirtsis G, Karydis M. 2001. Developing an optimal sampling design: A case study in coastal marine ecosystem. Environmental Monitoring and Assessment. 71:1-12.

Lembo G, Silecchia T, Carbonara P, Acrivulis A, Spedicato MT. 1998. A geostatistical approach to the assessment of the spatial distribution of *Parapenaeus longirostris* (Lucas, 1846) in the central-southern Tyrrhenian Sea. Crustaceana 72(9):1093-1095.

Mace PM. 1994. Relationships between common biological reference points used as thresholds and targets of fisheries management strategies. Canadian Journal of Fisheries and Aquatic Sciences 51:110-122.

Mace PM. 2001. A new Role for MSY in single-species and ecosystem approaches to fisheries stock assessment and management. Fish & Fisheries 2:2-32.

Maravelias CD, Reid DG, Simmonds EJ, Haralabous J. 1996. Spatial analysis and mapping of acoustic survey data in the presence of high local variability:

Geostatistical application to North Sea herring (*Clupea harengus*). Canadian Journal of Fisheries and Aquatic Sciences 53(07):1497-1505.

Maynou FX, Sarda F, Conan GY. 1998. Assessment of the spatial structure and biomass evaluation of *Nephrops norvegicus* (L.) populations in the northwestern Mediterranean by geostatistics. ICES Journal of Marine Science. 55(1):102-120.

McNaught DC, Steneck RS. 1998. Settlement and survival of the green sea urchin in Maine: Effects of algal habitat. Final report to the Maine Department of Marine Resources. West Boothbay Harbor, ME.

Pelletier D, Parma AM. 1994. Spatial distribution of Pacific halibut (*Hippoglossus stenolepis*): An application of geostatistics to longline survey data. Canadian Journal of Fisheries and Aquatic Sciences 51(7):1506-1518.

Petitgas P. 1993. Geostatistics for fish stock assessments: A review and an acoustic application. ICES Journal of Marine Science 50(3):285-298.

Petitgas P. 2001. Geostatistics in fisheries survey design and stock assessment: Models, variances and applications. Fish & Fisheries Series 2(3):231-249.

Peucker TK, Fowler RJ, Little JL, Mark DM 1976. Digital representation of three-dimensional surfaces by triangulated irregular networks. Office of Naval Research, Geography Programs. Arlington, VA, USA.

Quinn T, Deriso RB. 1999. Quantitative fish dynamics. Oxford, UK: Oxford University Press.

Restrepo VR. 1999. Providing scientific advice to implement the precautionary approach under the Magnuson-Steven Fishery Conservation and Management Act.

Proceedings of the Fifth National NMFS Stock Assessment Workshop NOAA Technical Memo, NMFS-F/SPO-40.

Ricker WE. 1975. Computation and interpretation of biological statistics of fish populations. Bulletin of the Fisheries Research Board of Canada 191.

Ripley BD. 1981. Spatial Statistics. New York: John Wiley & Sons. 252 p.

Rivard D, Maguire J-J. 1993. References points for fisheries management: eastern Canadian experience. In: Smith SJ, Hunt JJ, Rivard D, editors. Risk evaluation and biological reference points for fisheries management: Canadian Special Publication of Fisheries and Aquatic Science. p 31-57.

Rivoirard J, Simmonds J, Foote KG, Fernandes P, Bez N. 2000. Geostatistics for estimating fish abundance. Oxford: Blackwell Science. 206 p.

Rossi RE, Mulla DJ, Journel AG, Franz EH. 1992. Geostatistical Tools for Modeling and Interpreting Ecological Spatial Dependence. Ecological Monographs 62(2):277-314.

Roworth E, Signell R. 2002. Construction of a digital bathymetry for the Gulf of Maine. http://woodshole.er.usgs.gov/project-pages/oracles/gomaine/bathy/data.htm. 22 October 2002.

Scheibling RE, Hennigar AW, Balch T. 1999. Destructive grazing, epiphytism, and disease: the dynamics of sea urchin-kelp interactions in Nova Scotia. Canadian Journal of Fisheries and Aquatic Sciences 56(12):2300-2314.

Scheibling RE, Hatcher BG. 2001. The ecology of *Strongylocentrotus droebachiensis*. In: Lawrence JM, editor. Edible sea urchins: Biology and ecology. New York: Elsevier Science. p 271-306.

Simard Y, Legendre P, Lavoie G, Marcotte D. 1992. Mapping, estimating biomass, and optimizing sampling programs for spatially autocorrelated data: Case study of the northern shrimp (*Pandalus borealis*). Canadian Journal of Fisheries and Aquatic Sciences 49(1):32-45.

Smith SJ, Hunt JJ, Rivard D. 1993. Risk evaluation and biological reference points for fisheries management. Canadian Special Publication of Fisheries and Aquatic Science 120.

Vadas RL, Elner RW, Garwood PE, Babb IG. 1986. Experimental evaluation of aggregation behavior in the sea urchin *Strongylocentrotus droebachiensis*: A reinterpretation. Marine Biology 90:434-448.

Vadas RL, Steneck RS. 1995. Overfishing and inferences in kelp-sea urchin interactions. In: Skjodahl HR, Hopkins C, Erikstad KE, Leinaas, editors. Ecology of fjords and coastal waters. New York: Elsevier Science.

Vadas RL, Beal B, Dudgeon S, Wright W. 1997. Reproductive biology of green sea urchins on the coast of Maine. Final report to the Maine Department of Marine Resources. West Boothbay Harbor, ME.

Vadas RL, Sr., Smith BD, Beal B, Dowling T. 2002. Sympatric growth morphs and size bimodality in the green sea urchin (*Strongylocentrotus droebachiensis*). Ecological Monographs 72(1):113-132.

Vavrinec J, McNaught. 2001. Alternate stable states and the management of the green sea urchin *Strongylocentrotus droebachiensis* in Maine. Gulf of Mexico Science. 19(2):193.

van Groenigen JW. 2000. The influence of variogram parameters on optimal sampling schemes for mapping by kriging. Geoderma 97:223-236

Warren WG. 1998. Spatial analysis for marine populations: factors to be considered. In: Jamieson GS, Campbell A, editors. Proceedings of the North Pacific Symposium on Invertebrate Stock Assessment and Management: Canadian Special Publication of Fisheries and Aquatic Science p 21-28.

APPENDICES

Appendix A

C++ COMPUTER CODE FOR CALCULATING PROBABLISITIC ESTIMATES OF $F_{0.1}$ AND F_{MAX}

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<alloc.h>
#define SWAP(a, b) {float temp=(a);(a)=(b);(b)=temp;}
#define ROTATE(a, i, j, k, l) g=a[i][j]; h=a[k][l]; a[i][j]=(g-s*(h+g*tau));\
  a[k][l]=(h+s*(g-h*tau));
float gasdev(int *idum);
float ran1(int *idum);
void nrerror(char *error_text);
float *vector(int nl, int nh);
int *ivector(int nl, int nh);
void free_ivector(int *a, int nl, int nh);
void free_vector(float *a, int nl, int nh);
void gaussj(float **a, int n, float **b, int m);
float **matrix(int nrl, int nrh, int ncl, int nch);
void free_matrix(float **m, int nrl, int nrh, int ncl, int nch);
void ludcmp(float **a, int n, int *indx, float *d); /* SUBRUTINE FOR MATRIX*/
void lubksb(float **a, int n, int *indx, float *b);  /* INVERSE */
float lfunc(float *F);
float afunc(float *F);
float *W, *S, *dT, M, JA, L0, *Lt, *D, F;
float S_a, S_b, S_K, S_t0, S_L, S_m, S_L50, S_M, S_dis1, S_dis2;
int N2, A2, Index;
FILE *fp3;

void main()
{
 FILE *fp, *fp1, *fp2;
 int idum, N, A, k, i, j, B, BB;
 float *a, *b, m, L50, *K, *t0, *L, temp, d;
 float vL, va, vb, vm, vL50, vK, vt0, vM, Lt2;
 float x1, x2, xacc, dx, f, fmid, xmid, r, age_x;
 float dis1, dis2, vdis1, vdis2;
 int NN, NNN, I;
 int JMAX=40;
 /****Define parameter values*/
 d=1.0;
 idum=-100;
 xacc=0.0001;
 N=60;
 A=30;
 m=0.6676;
 L50=26.6;
 L0=40.0;
 dis1=0.3168;
 dis2=34.1;
 printf("Input the number of sets of simulated VBGF and W-L parameters\n");
 scanf("%d", &NN);
/*  printf("Input the number of sets of simulated W-L parameters\n");
 scanf("%d", &NNN);  */
 NNN=NN;
 /*****End***/
 K=vector(1, NN);
 L=vector(1, NN);
 t0=vector(1, NN);
 a=vector(1, NNN);
 b=vector(1, NNN);
 Lt=vector(1, N);
 S=vector(1, N);
 W=vector(1, N);
 dT=vector(1, N);
 D=vector(1, N);
 fp=fopen("sum", "w");
 fp3=fopen("test", "w");
 fp1=fopen("vbgf.dat", "r");
 fp2=fopen("wt.dat", "r");
 for(i=1; i<=NN; i++)
 fscanf(fp1, "%f  %f  %f\n", &L[i], &K[i], &t0[i]);
 for(i=1; i<=NNN; i++)
 fscanf(fp2, "%f  %f\n", &a[i], &b[i]);
 printf("Input the Natural mortality rate\n");
 scanf("%f", &M);
/* printf("Input errors for L50\n");
 scanf("%f", &vL50);
 printf("Input errors for m\n");
 scanf("%f", &vm);
 printf("Input errors for dis1\n");
 scanf("%f", &vdis1);
 printf("Input errors for dis2\n");
 scanf("%f", &vdis2);
*/
 vm=0.0;
 vL50=0.0;
 vdis1=0.0;
 vdis2=0.0;
 printf("Input error in natural mortality rate M\n");
 scanf("%f", &vM);
/* printf("Input number of simulation to be done\n");
 scanf("%d", &BB); */
 BB=NN;
 /***********start simulation********/
 for(B=1; B<=BB; B++)
 {
/*
  I=ran1(&idum)*NN+1;
  if(I>NN) I=NN;
*/
  S_K=K[B];
  S_L=L[B];
  S_t0=t0[B];
/*
  I=ran1(&idum)*NNN+1;
  if(I>NNN) I=NNN;
*/
  S_a=a[B];
  S_b=b[B];

printf("K=%6.3f L=%4.1f  t0=%4.3f  a=%4.3f  b=%4.3f\n", S_K, S_L, S_t0, S_a, S_b);

  for(i=1; i<=500; i++)
  {
   S_L50=L50*exp(vL50*gasdev(&idum));
   if(S_L50<(L50*1.3) && S_L50>(L50*0.7)) i=502;
   else printf("No suitable L50 was found in simulation run %d\n", B);
  }
  for(i=1; i<=500; i++)
  {
   S_m=m*exp(vm*gasdev(&idum));
   if(S_m<(m*1.3) && S_m>(m*0.7)) i=502;
   else printf("No suitable m was found in simulation run %d\n", B);
  }
  for(i=1; i<=500; i++)
  {
   S_M=M*exp(vM*gasdev(&idum));
   if(S_M<(M*1.3) && S_M>(M*0.7)) i=502;
   else printf("No suitable M was found in simulation run %d\n", B);
  }
  for(i=1; i<=500; i++)
  {
   S_dis1=dis1*exp(vdis1*gasdev(&idum));
  if(S_dis1<(dis1*1.3) && S_dis1>(dis1*0.7)) i=502;
  else printf("No suitable dis1 was found in simulation run %d\n", B);
 }
 for(i=1; i<=500; i++)
 {
  S_dis2=dis2*exp(vdis2*gasdev(&idum));
  if(S_dis2<(dis2*1.3) && S_dis2>(dis2*0.7)) i=502;
  else printf("No suitable dis2 was found in simulation run %d\n", B);
 }
 JA=S_t0-1/S_K*log(1.0/S_L);
 A2=JA;
 if(A2>A) A2=A;
 JA=S_L-L0-1.0;
 N2=JA;
 if(N2>N) N2=N;
 Lt[1]=L0;
 for(i=2; i<=N2; i++) Lt[i]=Lt[i-1]+1.0;

/*Calculate F0.1 from age-based YPR model*/
 for(Index=1; Index<=2; Index++)
 {

/*
  for(i=1; i<=A2; i++)
  {
   Lt2=S_L*(1.0-exp(-S_K*(i-S_t0)));
   S[i]=1.0/(1.0+exp(-S_m*(Lt2-S_L50)));
   if(Lt2>=50) S[i]=1.0;
   else S[i]=0;
   D[i]=1.0/(1.0+exp(S_dis1*(Lt2-S_dis2)));
   D[i]=0.0;
   W[i]=S_a*pow(Lt2, S_b);
  }
  x1=0;
  x2=10.0;
  fmid=afunc(&x2);
  f=afunc(&x1);
  if(f*fmid>0)
  {
   printf("Need to redefine x1 and x2\n");
   return;
  }
  if(f*fmid<0)
  {
   if(f<0)
   {
    r=x1;
    dx=x2-x1;
   }
   else
   {
    r=x2;
    dx=x1-x2;
   }
  }
  for(j=1; j<=JMAX; j++)
  {
   dx=dx*0.5;
   xmid=r+dx;
   fmid=afunc(&xmid);
   if(fmid<=0) r=xmid;
   if(fabs(dx)<xacc)
   {
    printf("dx=%f\n", fabs(dx));
    j=JMAX+1;
   }
   if(fmid==0) j=JMAX+1;
  }
  fprintf(fp, "%d    %5.4f  ", B, xmid);
  printf("PASS SIMULATION RUN %d USING AGE MODEL  F=%5.4f\n", B, xmid);
  age_x=xmid;

*/
 /***End of using age-structured model***/
/**Using length-structured model to estimate F0.1**/
  for(i=1; i<=N2; i++)
  {
   Lt2=Lt[i];
 /*  S[i]=1.0/(1.0+exp(-S_m*(Lt2-S_L50))); */
   if(Lt2>=50) S[i]=1.0;
   else S[i]=0.0;
   W[i]=S_a*pow(Lt2, S_b);
  }
  x1=0;
  x2=10.0;
  fmid=lfunc(&x2);
  f=lfunc(&x1);
/*
  if(f*fmid>0)
  {
   printf("Need to redefine x1 and x2\n");
   return;
   }
 */
   if(f*fmid<0)
   {
    if(f<0)
    {
     r=x1;
     dx=x2-x1;
    }
    else
    {
     r=x2;
     dx=x1-x2;
    }
   }
   for(j=1; j<=JMAX; j++)
   {
    dx=dx*0.5;
    xmid=r+dx;
    fmid=lfunc(&xmid);
    if(fmid<=0) r=xmid;
    if(fabs(dx)<xacc)
    {
     printf("dx=%f\n", fabs(dx));
     j=JMAX+1;
    }
    if(fmid==0) j=JMAX+1;
   }
   fprintf(fp, "%d   %5.4f   ", B, xmid);
   printf("PASS SIMULATION RUN %d USING LENGTH MODEL  F=%5.4f\n", B,
 xmid);
  }
  fprintf(fp, "\n");
 /***End of using length-structured model***/
 }
 fclose(fp);
 fclose(fp1);
 fclose(fp2);
 fclose(fp3);
 free_vector(K, 1, NN);
 free_vector(L, 1, NN);
 free_vector(t0, 1, NN);
 free_vector(a, 1, NNN);
 free_vector(b, 1, NNN);
 free_vector(S, 1, N);
 free_vector(W, 1, N);
 free_vector(dT, 1, N);
 free_vector(Lt, 1, N);
 free_vector(D, 1, N);
}

void nrerror(error_text)
char error_text[];
{ void exit();
  fprintf(stderr, "Run-time error...\n");
  fprintf(stderr, "%s\n", error_text);
  fprintf(stderr, "...now exiting to system...\n");
  exit(1);
}

float *vector(nl, nh)
int nl, nh;
{ float *v;
  v=(float *)malloc((unsigned)(nh-nl+1)*sizeof(float));
  if(!v) nrerror("allocation failure in vector()");
  return v-nl;
}

void free_vector(v, nl, nh)
float *v;
int nl, nh;
{
 free((char*) (v+nl));
}

void gaussj(a, n, b, m)
float **a, **b;
int n, m;
{
 int *indxc, *indxr, *ipiv;    /*three arrays are used for bookkeeping*/
 int i, icol, irow, j, k, l, ll, *ivector();      /*on the pivoting*/
 float big, dum, pivinv;
 void nrerror(), free_ivector();
 indxc=ivector(1,n);
 indxr=ivector(1,n);
 ipiv=ivector(1,n);
 for(j=1; j<=n; j++) ipiv[j]=0;
 for(i=1; i<=n; i++)
 {
  big=0.0;
  for(j=1; j<=n; j++)    /*outer loop of the search for a pivot element*/
   if(ipiv[j] !=1)
 for(k=1; k<=n; k++)
 {
  if(ipiv[k]==0)
  {
   if(fabs(a[j][k])>=big)
   {
    big=fabs(a[j][k]);
    irow=j;
    icol=k;
   }
  }
  else if (ipiv[k] > 1) nrerror("GAUSSJ: Singular Matrix-1");
 }
++(ipiv[icol]);
if(irow !=icol)
{
 for(l=1; l<=n; l++) SWAP(a[irow][l], a[icol][l])
 for(l=1; l<=m; l++) SWAP(b[irow][l], b[icol][l])
}
indxr[i]=irow; /*ready to divide the pivot row by the pivot element*/
indxc[i]=icol;        /*located at irow and icol*/
if(a[icol][icol]==0.0) nrerror("GAUSSJ: Singular Matrix-2");
pivinv=1.0/a[icol][icol];
a[icol][icol]=1.0;
for(l=1; l<=n; l++) a[icol][l] *=pivinv;
for(l=1; l<=m; l++) b[icol][l] *=pivinv;
for(ll=1; ll<=n; ll++)  /*reduce the row...*/
 if(ll !=icol)       /*except for the pivot one*/
 {
  dum=a[ll][icol];
  a[ll][icol]=0.0;
  for(l=1; l<=n; l++) a[ll][l]-=a[icol][l]*dum;
  for(l=1; l<=m; l++) b[ll][l]-=b[icol][l]*dum;
 }
 }
 for(l=n; l>=1; l--)
 {
  if(indxr[l] !=indxc[l])
   for(k=1; k<=n; k++)
   SWAP(a[k][indxr[l]], a[k][indxc[l]]);
  }
  free_ivector(ipiv, 1, n);
  free_ivector(indxr, 1, n);
  free_ivector(indxc, 1, n);
}
```

```
/*Gaussian noises*/
float gasdev(idum)
int *idum;
{
 static int iset=0;
 static float gset;
 float fac, r, v1, v2;
 float ran1();
 if(iset==0)
 {
  do
  {
   v1=2.0*ran1(idum)-1.0;
   v2=2.0*ran1(idum)-1.0;
   r=v1*v1+v2*v2;
  }
  while(r>=1.0);
  fac=sqrt(-2.0*log(r)/r);
  gset=v1*fac;
  iset=1;
  return v2*fac;
 }
 else
 {
  iset=0;
  return gset;
 }
}

/*generate random number*/

#define M1 259200
#define IA1 7141
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 243000
#define IA3 4561
#define IC3 51340

float ran1(idum)
int *idum;
{
 static long ix1, ix2, ix3;
 static float r[98];
 float temp;
 static int iff=0;
 int j;
 void nrerror();
 if(*idum<0 || iff==0)
  {
  iff=1;
  ix1=(IC1-(*idum)) % M1;
  ix1=(IA1*ix1+IC1) % M1;
  ix2=ix1 % M2;
  ix1=(IA1*ix1+IC1) % M1;
  ix3=ix1 % M3;
  for (j=1; j<=97; j++)
   {
   ix1=(IA1*ix1+IC1) % M1;
   ix2=(IA2*ix2+IC2) % M2;
   r[j]=(ix1+ix2*RM2)*RM1;
   }
  *idum=1;
 }
 ix1=(IA1*ix1+IC1) % M1;
 ix2=(IA2*ix2+IC2) % M2;
 ix3=(IA3*ix3+IC3) % M3;
 j=1 + ((97*ix3)/M3);
 if(j>97 || j<1) nrerror("ran1: this cannot happen.");
 temp=r[j];
 r[j]=(ix1+ix2*RM2)*RM1;
 return temp;
}

int *ivector(nl, nh)
int nl, nh;
{ int *v;
 v=(int *)malloc((unsigned)(nh-nl+1)*sizeof(int));
 if(!v) nrerror("allocation failure in ivector()");
 return v-nl;
}

void free_ivector(v, nl, nh)
int *v, nl, nh;
{
 free((char*) (v+nl));
}
```

```
float **matrix(nrl, nrh, ncl, nch)
int nrl, nrh, ncl, nch;
{ int i;
 float **m;
 m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float*));
 if(!m) nrerror("allocation failure 1 in matrix()");
 m-=nrl;

 for(i=nrl; i<=nrh; i++)
 {
  m[i]=(float *)malloc((unsigned) (nch-ncl+1)*sizeof(float));
  if (!m[i]) nrerror("allocation failure 2 in matrix()");
  m[i]-=ncl;
 }
 return m;
}

void free_matrix(m, nrl, nrh, ncl, nch)
float **m;
int nrl, nrh, ncl, nch;
{
 int i;
 for(i=nrh; i>=nrl; i--) free((char *) (m[i]+ncl));
 free((char *) (m+nrl));
}

float lfunc(F)
float *F;
{
 int i, k;
 float x1, x2, x3, temp1, temp2, temp3, temp, T1, T2, T3, T4, T5;
 x1=0.0;
 x2=0.0;
 for(i=1; i<=N2; i++)
 {
  temp1=0.0;
  temp2=0.0;
  temp3=log((S_L-Lt[i])/(S_L-Lt[i]-1))/S_K;
  for(k=1; k<i; k++)
  {
   temp=log((S_L-Lt[k])/(S_L-Lt[k]-1))/S_K;
   temp1+=S[k]*temp;
   temp2+=S_M*temp;
  }
  T1=S_M-(S[i]*(*F)*(*F)+(*F)*S_M)*temp1;
  T2=S_M-(S[i]*(*F)*(*F)+(*F)*S_M)*(temp1+S[i]*temp3);
  T3=(S[i]*(*F)+S_M)*(S[i]*(*F)+S_M);
  T4=-(*F)*temp1-temp2-(S[i]*(*F)+S_M)*temp3;
/*  T5=1.0-1.0/(1.0+exp(S_dis1*(Lt[i]-S_dis2)));*/
T5=1.0;
  x1+=T5*S[i]*W[i]*(T1*exp(-(*F)*temp1-temp2)-T2*exp(T4))/T3;
  if(Index==1)  x2+=T5*0.1*S[i]*W[i]*(1-exp(-S_M*temp3))*exp(-temp2)/S_M;
  else x2=0.0;
  }
 x3=x1-x2;
 return x3;
}

float afunc(F)
float *F;
{
 int i, k;
 float x1, x2, x3, temp1, temp2, T1, T2, T3, T4, T5;
 x1=0.0;
 x2=0.0;
 for(i=1; i<=A2; i++)
 {
  temp1=0.0;
  temp2=0.0;
  for(k=1; k<i; k++)
  {
   temp1+=S[k];
   temp2+=S_M;
  }
  T1=S_M-(S[i]*(*F)*(*F)+(*F)*S_M)*temp1;
  T2=S_M-(S[i]*(*F)*(*F)+(*F)*S_M)*(temp1+S[i]);
  T3=(S[i]*(*F)+S_M)*(S[i]*(*F)+S_M);
  T4=-(*F)*temp1-temp2-S[i]*(*F)-S_M;
/*  T5=1.0-D[i]; */
  T5=1.0;
  x1+=T5*S[i]*W[i]*(T1*exp(-(*F)*temp1-temp2)-T2*exp(T4))/T3;
  if(Index==1) x2+=T5*0.1*S[i]*W[i]*(1-exp(-S_M))*exp(-temp2)/S_M;
  else x2=0.0;

fprintf(fp3, "%d  %5.3f  %5.3f  %5.3f  %5.3f  %5.3f  %5.3f  %5.3f\n", i, temp1,
temp2,T1, T2, T3, T4, x1);


/*fprintf(fp3, "x1=%5.3f  x2=%5.3f\n ", x1, x2);*/

 }
 x3=x1-x2;
 return x3;
}
```

Appendix B

PROCEDURE AND COMPUTER CODE FOR ESTIMATING EXPLOITABLE BIOMASS

Procedure for estimating exploitable biomass using TINs

1. Create size (length) categories for the fish species. If the fishery has size restrictions, they should be considered when creating the size categories.
2. Calculate mean size for each category based on the size frequency information. Use a length-weight relationship to estimate the mean weight per urchin for each size category.
3. Using the size frequency information, calculate the proportion of fish at each site that belong to each size category.
4. Calculate mean densities by depth zone for each site
5. Scale each site's mean density value based on the size category proportions. So if you have 3 depth zones and 8 size categories, each site should have 24 different density values, which should sum to the total site average. Also, you will now have 24 different scenarios to run in order to estimate biomass.
6. Create text files with site locations and densities for each scenario.
7. Import each text file as a table into ArcView GIS 3.2a. ("Add table" command).
8. Add as an event theme to the view.
9. If the locations are not in the correct coordinate system, the data must be projected before you can continue. The following is one way to project data.
    a. Convert the text file to a shapefile

   b. Open the ArcToolbox Program in ArcINFO 7.1
   c. Define the coordinate system used in the density information
   d. Project the data to the correct coordinate system
   e. Add the projected shapefile back into ArcView GIS.

10. Choose the Extensions command under the File heading. Select 3D Analyst.
11. Activate the themes and select "Create TIN from features" under the Surface heading. Choose the following settings:
    a. Class=Point
    b. Height Source=Density
    c. Input as=Mass Points
    d. Value Field=<none>
12. Create Tins for all of the scenarios.
13. Convert TINs to Grids ("Convert to Grid" command under Theme heading)
14. Convert Grids to an ArcASCII format, using ArcToolbox
15. Repeat steps 9-14 to create ArcASCIIs for bathymetry and habitat. I recommend converting all of the files to Grids (step 11) at the same time, to ensure they all have the same resolution and orientation.
16. Once all ASCIIs are created, rename the urchin density ASCII to "gridA.asc," the habitat ASCII to "gridB.asc," and the bathymetry ASCII to "gridC.asc." Place these files in the same folder as the C++ biomass estimation program.
17. Run the C++ biomass estimation program. Follow the directions on the program. Note: Threshold values are density values used for estimating exploitable biomass. To estimate total or fishable biomass, enter a threshold value of 0.

18. The program will output the total number of urchins and the urchin biomass. An ASCII raster image will also be created for each scenario.

C++ Computer code for estimating biomass using ASCII files

biomass.cpp : calculates biomass of an arc ascii grid A based on constraints defined by arc ascii grids B and C.

```
#include "stdafx.h"
#include "biomass.h"
#include <fstream.h>
#include "math.h"
#include "Matrix.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// The one and only application object

CWinApp theApp;

//using namespace std;

int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
	int nRetCode = 0;

	// initialize MFC and print and error on failure
	if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
	{
		// TODO: change error code to suit your needs
		cerr << _T("Fatal Error: MFC initialization failed") << endl;
		return nRetCode = 1;
	}
/////////////////////////////////////////////////////////////////////////////
/////// MY CODE
```

```
	ifstream inFile;  // Input data file.
	ofstream outFile; // Output data file.
	CMatrix gridA, gridB, gridC;
	char chardummy;
	int gridAx, gridAy, gridBx, gridBy, gridCx, gridCy;
	int intdummy = 0, i, j, nodata, zone;
	double gridAres, gridBres, gridCres, doubledummy, biomass, count, weight,
minx, miny, coor, limit;


	cout << "Enter 1 to load gridA, gridB, and gridC\n";
	cin >> intdummy;


	gridA.Empty();    // open and read grids
	gridB.Empty();
	gridC.Empty();


	inFile.open("gridA.asc");
	if(!inFile)

	{
		cout << "Error opening file\n";
		return nRetCode;
	}

	// Prime gridA with data
	inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy;
	inFile >> gridAx;
	inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy;
	inFile >> gridAy;
	inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy;
	inFile >> minx;
	inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy;
	inFile >> miny;
	inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy;
	inFile >> gridAres;
```

```
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy;
        inFile >> nodata;

        gridA.SetMatrixSize(CSize (gridAx, gridAy));


        for (i=0;i<gridAy;i++)
                {
                        for (j=0;j<gridAx;j++)
                        {
                                inFile >> doubledummy;
                                gridA.SetAt(CPoint (j,i), doubledummy);
                        }
                }

        inFile.close();


        inFile.open("gridB.asc");
        if(!inFile)

        {
                cout << "Error opening file\n";
                return nRetCode;
        }

                // Prime gridB with data
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy;
        inFile >> gridBx;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy;
        inFile >> gridBy;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy;
        inFile >> coor;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy;
        inFile >> coor;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy;
        inFile >> gridBres;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy;
        inFile >> nodata;

        gridB.SetMatrixSize(CSize (gridBx, gridBy));


        for (i=0;i<gridBy;i++)
                {
                        for (j=0;j<gridBx;j++)
                        {
                                inFile >> doubledummy;
                                gridB.SetAt(CPoint (j,i), doubledummy);

                        }
                }

        inFile.close();


        inFile.open("gridC.asc");
        if(!inFile)

        {
                cout << "Error opening file\n";
                return nRetCode;
        }

        // Prime gridC with data
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy;
        inFile >> gridCx;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy;
        inFile >> gridCy;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy;
        inFile >> minx;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy;
        inFile >> miny;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy;
        inFile >> gridCres;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy;
        inFile >> nodata;

        gridC.SetMatrixSize(CSize (gridCx, gridCy));


        for (i=0;i<gridCy;i++)
                {
                        for (j=0;j<gridCx;j++)
                        {
                                inFile >> doubledummy;
                                gridC.SetAt(CPoint (j,i), doubledummy);
                        }
                }

        inFile.close();



        intdummy = 1;


        if ((gridAx != gridBx) || (gridAy != gridBy) || (gridAres != gridBres) || (gridAx !=
gridCx) || (gridAy != gridCy) || (gridAres != gridCres))
                {
                        cout << "\n" << "gridA, gridB, and gridC are not congruent!!!!\n"
<< "\n";
                }


        cout << "gridAx: " << gridAx << "   gridBx: " << gridBx << "   gridCx: " <<
gridCx << "\n";
        cout << "gridAy: " << gridAy << "   gridBy: " << gridBy << "   gridCy: " <<
gridCy << "\n";
        cout << "gridAres: " << gridAres << "   gridBres: " << gridBres << "   gridCres: "
<< gridCres << "\n";
        cout << "\n";
        cout << "Enter 1 to continue\n";
        cout << "Enter 2 to abort\n";
        cout << "Your choice: ";
        cin >> intdummy;
        cout << "\n";

        biomass = 0;
```

```
count = 0;

cout << "Please enter a threshold density: ";
cin >> limit;
cout << "\n";


outFile.open("output.asc");
if(!outFile)

{
        cout << "Error opening file\n";
        return nRetCode;
}

outFile << "NCOLS " << gridAx << "\n";
outFile << "NROWS " << gridAy << "\n";
outFile << "XLLCORNER " << minx << "\n";
outFile << "YLLCORNER " << miny << "\n";
outFile << "CELLSIZE " << gridAres << "\n";
outFile << "NODATA_VALUE " << nodata << "\n";


if (intdummy == 1)
{
        cout << "\n" << "Plese enter the weight of 1 urchin: ";
        cin >> weight;
        cout << "\n";
        cout << "Please make your depth choice:\n";
        zone = 0;
        cout << "1 ... 0 >= zone >= -5\n";
        cout << "2 ... -5 > zone >= -10\n";
        cout << "3 ... -10 > zone >= -15\n";
        cout << "4 ... -15 > zone >= -40\n";
        cin >> zone;
        cout << "\n";



        for (i=0;i<gridAy;i++)
                {
                        for (j=0;j<gridAx;j++)
                        {
                                if ((gridB.GetAt(CPoint (j,i))) != nodata)
                                {
                                        if ((gridA.GetAt(CPoint (j,i))) != nodata)
                                        {
                                            switch(zone)
                                            {
                                                case 1: if
((gridC.GetAt(CPoint (j,i)) <= 0) && (gridC.GetAt(CPoint (j,i)) >= -5))
                                                        {
                                                            if
(gridA.GetAt(CPoint (j,i)) >= limit)
                                                            {

    count += gridA.GetAt(CPoint (j,i));

    outFile << gridA.GetAt(CPoint (j,i));
                                                            }

                                                            else
                                                            {

    outFile << nodata;
                                                            }

                                                        }
                                                        else outFile
<< nodata;
                                                        break;

                                                case 2: if
((gridC.GetAt(CPoint (j,i)) < -5) && (gridC.GetAt(CPoint (j,i)) >= -10))
                                                        {
                                                            if
(gridA.GetAt(CPoint (j,i)) >= limit)
                                                            {

    count += gridA.GetAt(CPoint (j,i));

    outFile << gridA.GetAt(CPoint (j,i));
                                                            }

                                                            else
                                                            {

    outFile << nodata;
```

```
                                                                                                    }

                                                                                            }
                                                                                            else outFile
<< nodata;
                                                                                            break;


                                                                        case 3: if
((gridC.GetAt(CPoint (j,i)) < -10) && (gridC.GetAt(CPoint (j,i)) >= -15))
                                                                                            {
                                                                                                    if
(gridA.GetAt(CPoint (j,i)) >= limit)
                                                                                                    {

        count += gridA.GetAt(CPoint (j,i));

        outFile << gridA.GetAt(CPoint (j,i));
                                                                                                    }

                                                                                                    else
                                                                                                    {

        outFile << nodata;
                                                                                                    }
                                                                                            }
                                                                                            else outFile
<< nodata;
                                                                                            break;

                                                                        case 4: if
((gridC.GetAt(CPoint (j,i)) < -15) && (gridC.GetAt(CPoint (j,i)) >= -40))
                                                                                            {
                                                                                                    if
(gridA.GetAt(CPoint (j,i)) >= limit)
                                                                                                    {

        count += gridA.GetAt(CPoint (j,i));

        outFile << gridA.GetAt(CPoint (j,i));
                                                                                                    }

                                                                                                    else
                                                                                                    {
```

```
        outFile << nodata;
                                                                                         }


                                                                                 }
                                                                                 else outFile
<< nodata;
                                                                                 break;

                                                                case 5: if
((gridC.GetAt(CPoint (j,i)) <= 0) && (gridC.GetAt(CPoint (j,i)) >= -15))
                                                                                 {
                                                                                         if
(gridA.GetAt(CPoint (j,i)) >= limit)
                                                                                         {


        count += gridA.GetAt(CPoint (j,i));

        outFile << gridA.GetAt(CPoint (j,i));
                                                                                         }

                                                                                         else
                                                                                         {

        outFile << nodata;
                                                                                         }


                                                                                 }
                                                                                 else outFile
<< nodata;
                                                                                 break;

                                                                default: cout << "You can
only select one of the zones 1 to 5, try again!!!!!!\n";
                                                        }

                                                }

                                                else outFile << nodata;
                                        }

                                        else
                                        {
```

```
                        outFile << nodata;
                    }

                    outFile << " ";

                }

                outFile << "\n";
            }

        count = count * gridAres * gridAres;

        biomass = count * weight;

    }

    cout << "biomass: " << biomass << "\n";
    cout << "total # of urchins: " << count << "\n";
    cout << "resolution: " << gridAres << "\n";
    cout << "the raster overlay is stored in the file output.asc\n";
    cout << "enter 1 to finish\n";
    cin >> intdummy;

    return nRetCode;
}
```

Appendix C

PROCEDURE AND COMPUTER CODE FOR IDENTIFYING OPTIMAL SAMPLING STRATEGIES

Procedure for identifying optimal sampling strategies

1. Create a text file of fish densities by location. Place the x coordinate in the first column, the y coordinate in the second column and the density value in the third column. Do not include column headings in the text file.
2. Create an ArcASCII template file. This file indicates what regions have suitable habitat and potential fish abundance. The sampling program will be limited to these regions. Note: a buffer zone should be created around the region of interest in valid.asc. The width of the buffer zone should be equal to or greater than the size of the moving window (kernel).
    a. The file can be created directly in an ASCII format or it can be converted from other spatial formats, such as shapefiles, TINs, and grids, using the ArcToolbox program from ArcInfo 7.1.
3. Rename the urchin density text file "obs.txt" and the template ASCII to "valid.asc." and the bathymetry ASCII to "gridC.asc." Place these files in the same folder as the C++ kernel estimation program.
4. Run the C++ kernel estimation program. Follow the directions on the program. Note: The C++ code is designed for a stratified random strategy with a set number and size for the strata. The "Size of the moving window" is the kernel length and

is equivalent to the radius of a circle in pixels. We recommend limiting the number of simulations because the ArcASCII files can be very large.

5. When the program terminates, enter 1. Then run the C++ mean squared error (MSE) estimation program. The program will output the mean MSE and create an ASCII file of MSE.

C++ computer code for kernel estimation and implementation of a stratified random sampling strategy

```
// biomass.cpp : calculates biomass of an arc ascii grid A based on constraints
//
        definde by an arc ascii grid B.


#include "stdafx.h"
#include "biomass.h"
#include <fstream.h>
#include "math.h"
#include "Matrix.h"
#include "Location.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


        // Macro to get a random integer with a specified range
        #define getrandom(min, max) \
                ((rand()%(int)(((max) + 1)-(min)))+ (min))



CString int_to_string(int number, CString startstring)
{
        bool done = false;

        while (!done)
        {
                if ((number/10) < 1)
                {
                        if (number == 0)
                                startstring = "0" + startstring;
                        if (number == 1)
                                startstring = "1" + startstring;
                        if (number == 2)
                                startstring = "2" + startstring;
                        if (number == 3)
                                startstring = "3" + startstring;
                        if (number == 4)
                                startstring = "4" + startstring;
                        if (number == 5)
                                startstring = "5" + startstring;
                        if (number == 6)
                                startstring = "6" + startstring;
                        if (number == 7)
                                startstring = "7" + startstring;
                        if (number == 8)
                                startstring = "8" + startstring;
                        if (number == 9)
                                startstring = "9" + startstring;

                        done = true;
                }

                if ((number/10) >= 1)
                {
                        startstring = int_to_string((number-(int(number/10)*10)),
startstring);
                        number = int(number/10);
                }
        }

        return startstring;
}

/////////////////////////////////////////////////////////////////////////////
// The one and only application object

CWinApp theApp;

//using namespace std;

int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
```

```
{
        int nRetCode = 0;

        // initialize MFC and print and error on failure
        if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
        {
                // TODO: change error code to suit your needs
                cerr << _T("Fatal Error: MFC initialization failed") << endl;
                return nRetCode = 1;
        }
////////////////////////////////////////////////////////////////////////
//////// MY CODE


        ifstream inFile; // Input data file.
        ofstream outFile; // Output data file.
        CMatrix gridA, mastergridA, tempgrid, validgrid;
        char chardummy;
        int ncols, nrows, urxwin, urywin, xsample, ysample;
        int intdummy = 0, i, j, k, nodata=-9999, n, max=0, min=0, maxout, runs=0;
        double res, doubledummy, urchins, tau, kf, d;
        bool data;
        CLocation sample, tempobs;
        CList<CLocation,CLocation&> observations;
        CString filename, filename2;

        double llx, lly, llxwin, llywin, count;
        double winllx, winlly, winurx, winury, winarea;
        int windowsize, sampleloop;
        double wincenterx, wincentery;

        const pi=3.141592653589793;


        cout << "Enter 1 to load observations (obs.txt):";
        cin >> intdummy;
        cout << "\n";


        gridA.Empty();
        tempgrid.Empty();
        validgrid.Empty();
        mastergridA.Empty();


        inFile.open("obs.txt");
```

```
	if(!inFile)

	{
		cout << "Error opening file\n";
		return nRetCode;
	}

	while(inFile)
	{
		inFile >> sample.x >> sample.y >> sample.urchincount;
		if (inFile) observations.AddTail(sample);
	}

	inFile.close();


	cout << "Enter 1 to load the valid.asc grid outline (will be used for llx, lly, and
resolution): \n";
	cin >> intdummy;
	cout << " \n";

	inFile.open("valid.asc");
	if(!inFile)

	{
		cout << "Error opening file\n";
		return nRetCode;
	}

	inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy;
	inFile >> ncols;
	inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy;
	inFile >> nrows;
	inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy;
	inFile >> llx;
	inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy;
	inFile >> lly;
	inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy;
	inFile >> res;
```

```
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy;
        inFile >> nodata;

        validgrid.SetMatrixSize(CSize (ncols, nrows));


        for (i=0;i<nrows;i++)
                {
                        for (j=0;j<ncols;j++)
                        {
                                inFile >> doubledummy;
                                validgrid.SetAt(CPoint (j,i), doubledummy);
                        }
                }

        inFile.close();


        gridA.SetMatrixSize(CSize (ncols,nrows));
        tempgrid.SetMatrixSize(CSize (ncols,nrows));
        mastergridA.SetMatrixSize(CSize (ncols,nrows));

        cout << "Please enter the size of the moving window (half the side in pixel): \n";
        cin >> windowsize;
        cout << "\n";

        tau = (windowsize*res) + (res*0.5);

        winarea = ((2*windowsize*res + res) * (2*windowsize*res + res));


        for (i=0;i<nrows;i++)
        {
                for (j=0;j<ncols;j++)
                {
                        gridA.SetAt(CPoint (j,i),nodata);
                }
        }


        POSITION pos = observations.GetHeadPosition();

        for (i=windowsize;i<(nrows-windowsize);i++)
        {
                for (j=windowsize;j<(ncols-windowsize);j++)
                {

                        urchins = 0;
                        count = 0;

                        wincenterx = llx + j*res + 0.5*res;
                        wincentery = lly + nrows*res - (i*res + 0.5*res);
                        winllx = wincenterx - (0.5*res + windowsize*res);
                        winlly = wincentery - (0.5*res + windowsize*res);
                        winurx = wincenterx + (0.5*res + windowsize*res);
                        winury = wincentery + (0.5*res + windowsize*res);

                        pos = observations.GetHeadPosition();

                        for (k=0;k<observations.GetCount();k++)
                        {
                                tempobs = observations.GetNext(pos);

                                if ((tempobs.x >= winllx) && (tempobs.x < winurx) &&
(tempobs.y >= winlly) && (tempobs.y < winury))
                                {
                                        d = sqrt(((wincenterx - tempobs.x)*(wincenterx -
tempobs.x)) + ((wincentery - tempobs.y)*(wincentery - tempobs.y)));

                                        if (d <= tau)
                                        {
                                                kf = (3/pi)*((1-((d/tau)*(d/tau)))*(1-
((d/tau)*(d/tau))));

                                                urchins = urchins + (kf *
tempobs.urchincount);
                                                count = count + kf;


                                        }
                                }
                        }

                        if ((validgrid.GetAt(CPoint (j,i)) != nodata))
                        {
                                if (count == 0)
                                        gridA.SetAt(CPoint (j,i),nodata);

                                else
                                        gridA.SetAt(CPoint (j,i),(urchins/count));
                        }

                }
        }



        cout << "Number of Samples per Strata: ";
        cin >> n;
        cout << "\n";

        cout << "Number of runs is set to 20";
        runs = 20;
        cout << "\n";

        for (i=0;i<nrows;i++)
        {
                for (j=0;j<ncols;j++)
                {
                        mastergridA.SetAt(CPoint (j,i), (gridA.GetAt(CPoint (j,i))));
                }
        }


        for (sampleloop=0;sampleloop<runs;sampleloop++)
        {
                filename = "";
                filename = int_to_string((sampleloop+1),filename);
                filename = "sample_set" + filename + ".txt";


                for (i=0;i<nrows;i++)
                {
                        for (j=0;j<ncols;j++)
                        {
                                gridA.SetAt(CPoint (j,i), (mastergridA.GetAt(CPoint
(j,i))));
                        }
                }


                outFile.open(filename);
                if(!outFile)
                {
                        cout << "Error opening file\n";
                        return nRetCode;
```

```
                    }


                    //Zone 1

                    llxwin = int((362383.06-llx)/res);
                    llywin = (nrows - int((4768863.7-lly)/res));
                    urxwin = int((431450.46-llx)/res);
                    urywin = (nrows - int((4866671.22-lly)/res));

                    for (i=0;i<n;i++)
                    {
                            data = false;
                            maxout = 1000 * ((urxwin-llxwin)*(urywin-llywin));

                            while (!data)
                            {

                                    xsample = getrandom(llxwin,urxwin);
                                    ysample = getrandom(urywin,llywin);


                                    if (gridA.GetAt(CPoint (xsample,ysample)) != nodata)
                                    {
                                            outFile << (xsample*res+llx) << "\t" << ((nrows -
ysample)*res+lly) << "\t";
                                            outFile << gridA.GetAt(CPoint (xsample,ysample))
<< "\n";
                                            gridA.SetAt(CPoint (xsample,ysample),
double(nodata));

                                            data = true;
                                    }


                                    maxout = (maxout + 1);


                                    if (maxout == 0)
                                    {
                                            data = true;
                                            cout << "Zone 1 did not contain enough valid data
points!!! \n";

                                            i = n;
                                    }
```

```
                }
        }


        //Zone 2

        llxwin = int((431450.46-llx)/res);
        llywin = (nrows - int((4833354.37-lly)/res));
        urxwin = int((469027.7-llx)/res);
        urywin = (nrows - int((4883055.12-lly)/res));


        for (i=0;i<n;i++)
        {
                data = false;
                maxout = 200000 * ((urxwin-llxwin)*(urywin-llywin));

                while (!data)
                {
                        xsample = getrandom(llxwin,urxwin);
                        ysample = getrandom(urywin,llywin);


                        if (gridA.GetAt(CPoint (xsample,ysample)) != nodata)
                        {
                                outFile << (xsample*res+llx) << "\t" << ((nrows -
ysample)*res+lly) << "\t";
                                outFile << gridA.GetAt(CPoint (xsample,ysample))
<< "\n";
                                gridA.SetAt(CPoint (xsample,ysample),
double(nodata));

                                data = true;
                        }

                        maxout = (maxout + 1);

                        if (maxout == 0)
                        {
                                data = true;
                                cout << "Zone 2 did not contain enough valid data
points!!! \n";
                                i = n;
                        }
                }
```

```
	}

	//Zone 3

	llxwin = int((469027.70-llx)/res);
	llywin = (nrows - int((4849736.33-lly)/res));
	urxwin = int((499998.65-llx)/res);
	urywin = (nrows - int((4916303.1-lly)/res));


	for (i=0;i<n;i++)
	{
		data = false;
		maxout = 1000 * ((urxwin-llxwin)*(urywin-llywin));

		while (!data)
		{
			xsample = getrandom(llxwin,urxwin);
			ysample = getrandom(urywin,llywin);

			if (gridA.GetAt(CPoint (xsample,ysample)) != nodata)
			{
				outFile << (xsample*res+llx) << "\t" << ((nrows -
ysample)*res+lly) << "\t";
				outFile << gridA.GetAt(CPoint (xsample,ysample))
<< "\n";
				gridA.SetAt(CPoint (xsample,ysample),
double(nodata));

				data = true;
			}

			maxout = (maxout + 1);

			if (maxout == 0)
			{
				data = true;
				cout << "Zone 3 did not contain enough valid data
points!!! \n";
				i = n;
			}
		}
	}

	//Zone 4
```

```
		llxwin = int((499998.65-llx)/res);
		llywin = (nrows - int((4866320.69-lly)/res));
		urxwin = int((535269.05-llx)/res);
		urywin = (nrows - int((4940833.53-lly)/res));


		for (i=0;i<n;i++)
		{
			data = false;
			maxout = 1000 * ((urxwin-llxwin)*(urywin-llywin));

			while (!data)
			{
				xsample = getrandom(llxwin,urxwin);
				ysample = getrandom(urywin,llywin);

				if (gridA.GetAt(CPoint (xsample,ysample)) != nodata)
				{
					outFile << (xsample*res+llx) << "\t" << ((nrows -
ysample)*res+lly) << "\t";
					outFile << gridA.GetAt(CPoint (xsample,ysample))
<< "\n";
					gridA.SetAt(CPoint (xsample,ysample),
double(nodata));

					data = true;
				}

				maxout = (maxout + 1);

				if (maxout == 0)
				{
					data = true;
					cout << "Zone 4 did not contain enough valid data
points!!! \n";
					i = n;
				}
			}
		}

		//Zone 5

		llxwin = int((535269.05-llx)/res);
		llywin = (nrows - int((4871969.51-lly)/res));
		urxwin = int((561878.3-llx)/res);
		urywin = (nrows - int((4940833.53-lly)/res));
```

```
			for (i=0;i<n;i++)
			{
				data = false;
				maxout = 1000 * ((urxwin-llxwin)*(urywin-llywin));

				while (!data)
				{
					xsample = getrandom(llxwin,urxwin);
					ysample = getrandom(urywin,llywin);

					if (gridA.GetAt(CPoint (xsample,ysample)) != nodata)
					{
						outFile << (xsample*res+llx) << "\t" << ((nrows -
ysample)*res+lly) << "\t";
						outFile << gridA.GetAt(CPoint (xsample,ysample))
<< "\n";
						gridA.SetAt(CPoint (xsample,ysample),
double(nodata));

						data = true;
					}

					maxout = (maxout + 1);

					if (maxout == 0)
					{
						data = true;
						cout << "Zone 5 did not contain enough valid data
points!!! \n";
						i = n;
					}
				}
			}

			//Zone 6

			llxwin = int((561878.3-llx)/res);
			llywin = (nrows - int((4905491.63-lly)/res));
			urxwin = int((587744.29-llx)/res);
			urywin = (nrows - int((4940833.53-lly)/res));


			for (i=0;i<n;i++)
			{
```

```
                    data = false;
                    maxout = 1000 * ((urxwin-llxwin)*(urywin-llywin));

                    while (!data)
                    {
                         xsample = getrandom(llxwin,urxwin);
                         ysample = getrandom(urywin,llywin);

                         if (gridA.GetAt(CPoint (xsample,ysample)) != nodata)
                         {
                              outFile << (xsample*res+llx) << "\t" << ((nrows -
ysample)*res+lly) << "\t";
                              outFile << gridA.GetAt(CPoint (xsample,ysample))
<< "\n";
                              gridA.SetAt(CPoint (xsample,ysample),
double(nodata));

                              data = true;
                         }

                         maxout = (maxout + 1);

                         if (maxout == 0)
                         {
                              data = true;
                              cout << "Zone 6 did not contain enough valid data
points!!! \n";
                              i = n;
                         }
                    }
               }

               //Zone 7

               llxwin = int((587744.29-llx)/res);
               llywin = (nrows - int((4905784.11-lly)/res));
               urxwin = int((617253.62-llx)/res);
               urywin = (nrows - int((4950690.77-lly)/res));


               for (i=0;i<n;i++)
               {
                    data = false;
                    maxout = 1000 * ((urxwin-llxwin)*(urywin-llywin));

                    while (!data)
```

```
			{
				xsample = getrandom(llxwin,urxwin);
				ysample = getrandom(urywin,llywin);

				if (gridA.GetAt(CPoint (xsample,ysample)) != nodata)
				{
					outFile << (xsample*res+llx) << "\t" << ((nrows -
ysample)*res+lly) << "\t";
					outFile << gridA.GetAt(CPoint (xsample,ysample))
<< "\n";
					gridA.SetAt(CPoint (xsample,ysample),
double(nodata));

					data = true;
				}

				maxout = (maxout + 1);

				if (maxout == 0)
				{
					data = true;
					cout << "Zone 7 did not contain enough valid data
points!!! \n";
					i = n;
				}
			}
		}

		//Zone 8

		llxwin = int((617253.62-llx)/res);
		llywin = (nrows - int((4917368.44-lly)/res));
		urxwin = int((662104.01-llx)/res);
		urywin = (nrows - int((4963889.58-lly)/res));


		for (i=0;i<n;i++)
		{
			data = false;
			maxout = 1000 * ((urxwin-llxwin)*(urywin-llywin));

			while (!data)
			{
				xsample = getrandom(llxwin,urxwin);
				ysample = getrandom(urywin,llywin);
```

```
                    if (gridA.GetAt(CPoint (xsample,ysample)) != nodata)
                    {
                         outFile << (xsample*res+llx) << "\t" << ((nrows -
ysample)*res+lly) << "\t";
                         outFile << gridA.GetAt(CPoint (xsample,ysample))
<< "\n";
                         gridA.SetAt(CPoint (xsample,ysample),
double(nodata));

                         data = true;
                    }

                    maxout = (maxout + 1);

                    if (maxout == 0)
                    {
                         data = true;
                         cout << "Zone 8 did not contain enough valid data
points!!! \n";
                         i = n;
                    }
               }
          }

          //Zone 9

          llxwin = int((637903.62-llx)/res);
          llywin = (nrows - int((4963889.58-lly)/res));
          urxwin = int((662104.01-llx)/res);
          urywin = (nrows - int((4985552.27-lly)/res));


          for (i=0;i<n;i++)
          {
               data = false;
               maxout = 1000 * ((urxwin-llxwin)*(urywin-llywin));

               while (!data)
               {
                    xsample = getrandom(llxwin,urxwin);
                    ysample = getrandom(urywin,llywin);

                    if (gridA.GetAt(CPoint (xsample,ysample)) != nodata)
                    {
                         outFile << (xsample*res+llx) << "\t" << ((nrows -
ysample)*res+lly) << "\t";
```

```
                                        outFile << gridA.GetAt(CPoint (xsample,ysample))
<< "\n";
                                        gridA.SetAt(CPoint (xsample,ysample),
double(nodata));

                                        data = true;
                                }

                                maxout = (maxout + 1);

                                if (maxout == 0)
                                {
                                        data = true;
                                        cout << "Zone 9 did not contain enough valid data
points!!! \n";
                                        i = n;
                                }
                        }
                }

                outFile.close();

        }



        for (sampleloop=0;sampleloop<runs;sampleloop++)
        {
                filename = "";
                filename = int_to_string((sampleloop+1),filename);
                filename = "mean_result" + filename + ".asc";


                filename2 = "";
                filename2 = int_to_string((sampleloop+1),filename2);
                filename2 = "sample_set" + filename2 + ".txt";


                while (!observations.IsEmpty())
                {
                        observations.RemoveTail();
                }

                for (i=0;i<nrows;i++)
                {
                        for (j=0;j<ncols;j++)
```

```
		{
			tempgrid.SetAt (CPoint (j,i),nodata);
		}
	}


	inFile.open(filename2);
	if(!inFile)

	{
		cout << "Error opening file\n";
		return nRetCode;
	}

	while(inFile)
	{
		inFile >> sample.x >> sample.y >> sample.urchincount;
		if (inFile) observations.AddTail(sample);
	}

	inFile.close();


	POSITION pos = observations.GetHeadPosition();

	for (i=windowsize;i<(nrows-windowsize);i++)
	{
		for (j=windowsize;j<(ncols-windowsize);j++)
		{

			urchins = 0;
			count = 0;

			wincenterx = llx + j*res + 0.5*res;
			wincentery = lly + nrows*res - (i*res + 0.5*res);
			winllx = wincenterx - (0.5*res + windowsize*res);
			winlly = wincentery - (0.5*res + windowsize*res);
			winurx = wincenterx + (0.5*res + windowsize*res);
			winury = wincentery + (0.5*res + windowsize*res);

			pos = observations.GetHeadPosition();

			for (k=0;k<observations.GetCount();k++)
			{
				tempobs = observations.GetNext(pos);
```

```
                                        if ((tempobs.x >= winllx) && (tempobs.x < winurx)
&& (tempobs.y >= winlly) && (tempobs.y < winury))
                                        {
                                                d = sqrt(((wincenterx -
tempobs.x)*(wincenterx - tempobs.x)) + ((wincentery - tempobs.y)*(wincentery -
tempobs.y)));

                                                if (d <= tau)
                                                {
                                                        kf = (3/pi)*((1-((d/tau)*(d/tau)))*(1-
((d/tau)*(d/tau))));

                                                        urchins = urchins + (kf *
tempobs.urchincount);

                                                        count = count + kf;
                                                }
                                        }
                                }

                                if ((validgrid.GetAt(CPoint (j,i)) != nodata))
                                {
                                        if (count == 0)
                                                tempgrid.SetAt(CPoint (j,i),nodata);

                                        else
                                                tempgrid.SetAt(CPoint (j,i),(urchins/count));
                                }

                        }
                }


                outFile.open(filename);
                if(!outFile)
                {
                        cout << "Error opening file\n";
                        return nRetCode;
                }

                outFile << "NCOLS " << ncols << "\n";
                outFile << "NROWS " << nrows << "\n";
                outFile << "XLLCORNER " << llx << "\n";
                outFile << "YLLCORNER " << lly << "\n";
                outFile << "CELLSIZE " << res << "\n";
                outFile << "NODATA_VALUE " << nodata << "\n";
                for (i=0;i<nrows;i++)
                        {
                                for (j=0;j<ncols;j++)
                                {
                                        outFile << tempgrid.GetAt(CPoint (j,i));
                                        outFile << " ";
                                }

                                outFile << "\n";
                }

                outFile.close();

        }


        cout << "\n";
        cout << "The output is stored in 40 files: \n" << "    20 with sample locations and
20 resulting averages";
        cout << "\n";
        cout << "enter 1 to finish\n";
        cin >> intdummy;

        return nRetCode;
}
```

<u>C++ computer code for generating plots of mean squared error (MSE) and mean MSE</u>

```
//Calculates MSE.  Arc ASCII gridA is the original density file, arc ASCII tempgrid is
//the simulated density file, and arc ASCII grid B is the depth and habitat constraints


#include "stdafx.h"
#include "biomass.h"
#include <fstream.h>
#include "math.h"
#include "Matrix.h"
#include "Location.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
```

```
#endif



CString int_to_string(int number, CString startstring)
{
        bool done = false;

        while (!done)
        {
                if ((number/10) < 1)
                {
                        if (number == 0)
                                startstring = "0" + startstring;
                        if (number == 1)
                                startstring = "1" + startstring;
                        if (number == 2)
                                startstring = "2" + startstring;
                        if (number == 3)
                                startstring = "3" + startstring;
                        if (number == 4)
                                startstring = "4" + startstring;
                        if (number == 5)
                                startstring = "5" + startstring;
                        if (number == 6)
                                startstring = "6" + startstring;
                        if (number == 7)
                                startstring = "7" + startstring;
                        if (number == 8)
                                startstring = "8" + startstring;
                        if (number == 9)
                                startstring = "9" + startstring;

                        done = true;
                }

                if ((number/10) >= 1)
                {
                        startstring = int_to_string((number-(int(number/10)*10)),
 startstring);
                        number = int(number/10);
                }
        }

        return startstring;
```

```
}


/////////////////////////////////////////////////////////////////////////////
// The one and only application object

CWinApp theApp;

//using namespace std;

int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
	int nRetCode = 0;

	// initialize MFC and print and error on failure
	if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
	{
		// TODO: change error code to suit your needs
		cerr << _T("Fatal Error: MFC initialization failed") << endl;
		return nRetCode = 1;
	}
///////////////////////////////////////////////////////////////////////////////
//////// MY CODE


	ifstream inFile; // Input data file.
	ofstream outFile; // Output data file.
	CMatrix gridA, tempgrid, result;
	char chardummy;
	int ncols, nrows;
	int intdummy = 0, i, j, nodata=-9999;
	double res, doubledummy, llx, lly;
	CString filename;
	int sampleloop;
	double a, b, c;


	gridA.Empty();
	tempgrid.Empty();

	int runs = 2;

	cout << "Enter 1 to load gridA.asc (will be used for llx, lly, and resolution): \n";
	cin >> intdummy;
	cout << " \n";
```

```
        inFile.open("gridA.asc");
        if(!inFile)

        {
                cout << "Error opening file\n";
                return nRetCode;
        }

        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy;
        inFile >> ncols;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy;
        inFile >> nrows;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy;
        inFile >> llx;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy;
        inFile >> lly;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy;
        inFile >> res;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy;
        inFile >> nodata;

        gridA.SetMatrixSize(CSize (ncols, nrows));


        for (i=0;i<nrows;i++)
                {
                        for (j=0;j<ncols;j++)
                        {
                                inFile >> doubledummy;
                                gridA.SetAt(CPoint (j,i), doubledummy);
                        }
                }

        inFile.close();


        tempgrid.SetMatrixSize(CSize (ncols,nrows));
        result.SetMatrixSize(CSize (ncols,nrows));
```

```
	for (i=0;i<nrows;i++)
			{
				for (j=0;j<ncols;j++)
				{
					result.SetAt(CPoint (j,i), 0.0);
				}
			}


	for (sampleloop=0;sampleloop<runs;sampleloop++)
	{

		filename = "";
		filename = int_to_string((sampleloop+1),filename);
		filename = "mean_result" + filename + ".asc";


		inFile.open(filename);
		if(!inFile)

		{
			cout << "Error opening file\n";
			return nRetCode;
		}

		inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy;
		inFile >> intdummy;
		inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy;
		inFile >> intdummy;
		inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy;
		inFile >> doubledummy;
		inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy;
		inFile >> doubledummy;
		inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy;
		inFile >> doubledummy;
		inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy;
		inFile >> intdummy;
		for (i=0;i<nrows;i++)
			{
				for (j=0;j<ncols;j++)
				{
					inFile >> doubledummy;
					tempgrid.SetAt(CPoint (j,i), doubledummy);
				}
			}

		inFile.close();



		for (i=0;i<nrows;i++)
			{
				for (j=0;j<ncols;j++)
				{
					if (tempgrid.GetAt(CPoint (j,i)) != nodata)
					{
						a = tempgrid.GetAt(CPoint (j,i));
						b = gridA.GetAt(CPoint (j,i));
						c = (b-a)*(b-a);
						c = c + result.GetAt(CPoint (j,i));
						result.SetAt(CPoint (j,i), c);
					}
				}
			}

	}

	for (i=0;i<nrows;i++)
			{
				for (j=0;j<ncols;j++)
				{
					if (result.GetAt(CPoint (j,i)) != nodata)
					{
						a = (result.GetAt(CPoint (j,i))/runs);
						result.SetAt(CPoint (j,i), a);
					}
				}
			}


	outFile.open("output.asc");
	if(!outFile)
	{
		cout << "Error opening file\n";
		return nRetCode;
	}

	outFile << "NCOLS " << ncols << "\n";
	outFile << "NROWS " << nrows << "\n";
	outFile << "XLLCORNER " << llx << "\n";
	outFile << "YLLCORNER " << lly << "\n";
	outFile << "CELLSIZE " << res << "\n";
	outFile << "NODATA_VALUE " << nodata << "\n";


	for (i=0;i<nrows;i++)
		{
			for (j=0;j<ncols;j++)
			{
				outFile << result.GetAt(CPoint (j,i));
				outFile << " ";
			}

			outFile << "\n";
		}

	outFile.close();


	cout << "\n";
	cout << "The output is stored in output.asc";
	cout << "\n";
	cout << "enter 1 to finish\n";
	cin >> intdummy;

	return nRetCode;
}


/*

	tempgrid.Empty();	// open and read grid


	inFile.open("somegrid.txt");
	if(!inFile)
```

```
        {
                cout << "Error opening file\n";
                return nRetCode;
        }

        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy;
        inFile >> intdummy;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy;
        inFile >> intdummy;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy;
        inFile >> doubledummy;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy;
        inFile >> doubledummy;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy;
        inFile >> doubledummy;
        inFile >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy >> chardummy >> chardummy >>
chardummy >> chardummy >> chardummy;
        inFile >> intdummy;


        for (i=0;i<nrows;i++)
                {
                        for (j=0;j<ncols;j++)
                        {
                                inFile >> doubledummy;
                                tempgrid.SetAt(CPoint (j,i), doubledummy);
                        }
                }

        inFile.close();


*/
```

BIOGRAPHY OF THE AUTHOR

Robert Grabowski was born in Chicago, IL on 17 August 1978. His parents raised him and his older brother and sister in their home in Niles, IL. Robert graduated from Maine Township High School East, in Park Ridge, IL, in May 1996. He went on to earn a Bachelor of Science degree in Biology from Illinois Wesleyan University in April 2000. After university, Robert worked as a research technician at the Centre for Rainforest Studies in Yungaburra, Australia. He spent a year studying rainforest restoration, teaching bird ecology and researching leaf litter detritivores. Robert moved back to the US for graduate studies at the University of Maine's School of Marine Sciences in September 2001. He is a candidate for The Master of Science degree in Marine Biology from The University of Maine in May, 2003.