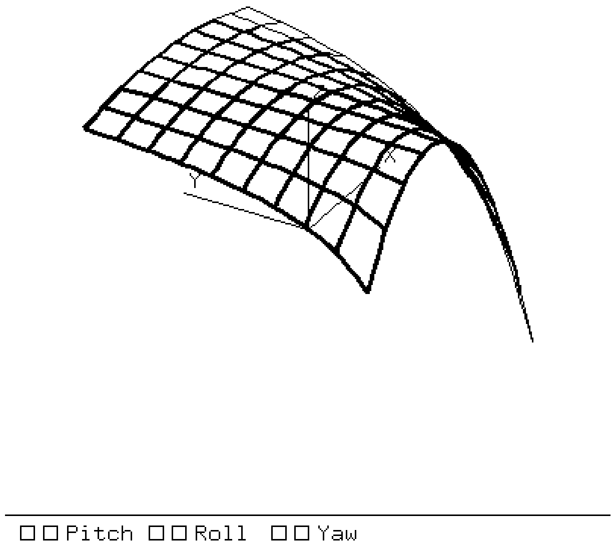
Introduction to Lisp-Stat and its applications.

Suggested Solutions to Exercises 2.

1.
```
(defun log-lik-norm (data)
   #'(lambda (mu sigma)
        (- (- (/ (sum (^ (- data mu) 2)) 2 (^ sigma 2)))
  (* 0.5 (length data) (log (* 2 pi sigma sigma))))))

(def llnorm-gas1 (log-lik-norm gas1))

(spin-function llnorm-gas1 0 40 1 3 :num-points 10)
```




```
(newtonmax #'(lambda (theta)
       (funcall llnorm-gas1 (first theta) (second theta)))
   '(14 1))
;; maximizing...
;; Iteration 0.
;; Criterion value = -137.807
;; Iteration 1.
;; Criterion value = -112.126
;; Iteration 2.
;; Criterion value = -103.203
;; Iteration 3.
```

```
;; Criterion value = -100.913
;; Iteration 4.
;; Criterion value = -100.640
;; Iteration 5.
;; Criterion value = -100.634
;; Iteration 6.
;; Criterion value = -100.634
;; Reason for termination: gradient size is less than gradient tolerance.
;; (14.628999923949154 1.810735094391804)

(list (mean gas1) (* (sqrt (/ 49 50)) (standard-deviation gas1)))
;; (14.629000000000001 1.8107327246173028)
```

2.

```
(defun anova (data)
  (if (not (and data
(listp data)
(reduce #'(lambda (x y) (and x y))
(mapcar #'listp data))
(reduce #'(lambda (x y) (and x y))
(mapcar #'numberp (reduce #'append data)))))
      (error "Data is not in correct form"))
  (let* ((n (mapcar #'length data))
 (means (mapcar #'mean data))
 (overall-mean (mean data)))
    (list (sum (^ (- data means) 2))
  (sum (* n (^ (- means overall-mean) 2))))))

(def blood '((62 60 63 59) (63 67 71 64 65 66) (68 66 71 67 68 68)
               (56 62 60 61 63 64 63 59)))
(anova blood)
;; (112.0 228.0)
```

3. (a)
```
(load "/usr/people59/canty/teach/epfl/lispstat/datasets")
(defmeth one-sample-proto :t-test (&optional (mu0 0))
  (let* ((data (slot-value 'data))
 (xbar (mean data))
 (s (standard-deviation data))
 (n (length data))
 (t-stat (/ (- xbar mu0) (/ s (sqrt n))))
 (p-val (* 2 (- 1 (t-cdf (abs t-stat) (- n 1))))))
    (format t "Students t-test of H0: mu=~,2g for ~a~%"
    mu0 (if (slot-value 'title) (slot-value 'title) "a dataset"))
    (format t "~%The t-statistic is ~,4f on ~d degrees of freedom.~%"
    t-stat (- n 1))
    (format t "The p-value is ~,4f~%" p-val)))

(def shot-put '(17.79 17.21 16.47 16.27 15.53 14.92 14.72 14.63
14.23 13.70 14.52 14.46 13.22 11.75 16.31 15.81 14.98 14.30 14.05 13.66))

(def shot-put1 (send one-sample-proto :new shot-put
                     :title "Distances of Shot-Put Throws"
                     :doc
  "   The data set records the distances of throws in the men's shot-put
   at the 1991 English athletics championships."))

(send shot-put1 :t-test)
;; Students t-test of H0: mu=0.00 for Distances of Shot-Put Throws
;;
;; The t-statistic is 46.63 on 19 degrees of freedom.
;; The p-value is 0.0000

(send shot-put1 :t-test 15)
;; Students t-test of H0: mu=15. for Distances of Shot-Put Throws
;;
;; The t-statistic is -0.2296 on 19 degrees of freedom.
;; The p-value is 0.8208
```

(b)
```
(def ttest-item
     (send menu-item-proto :new "t test"
     :action #'(lambda ()
 (let ((obj (get-object)))
   (if (and obj
    (one-sample-p
       (send obj :slot-value 'data)))
       (send obj :t-test
          (first (get-value-dialog
   "Value of Mu_0")))
       (message-dialog "Not a Univariate Dataset"))
   (top-level nil)))))
(send data-menu :append-items ttest-item)
```

(c)
```
(defproto paired-data-proto () () data-set-proto)

(defmeth paired-proto :isnew (data &key title doc)
  (if (and (two-sample-p data)
   (= (length (first data)) (length (second data))))
      (call-next-method :data data :title title :info doc)
      (print "Data is not a paired data set.  Object not created")))

(defun make-dataset (data &key title doc (paired nil))
; We add the paired keyword because it is possible for a two-sample
; data set to have both samples the same length but not be paired.
   (cond ((one-sample-p data)
          (send one-sample-proto :new
                :data data
                :title title
                :info doc))
 ((and paired
       (two-sample-p data)
       (= (length (first data)) (length (second data))))
  (send paired-proto :new
:data data
                :title title
                :info doc))
```

```
        ((two-sample-p data)
         (send two-sample-proto :new
               :data data
               :title title
               :info doc))
        (t (print "Structure not implemented"))))
```

(d)
```
(defmeth paired-proto :summary ()
  (let ((diffs (reduce #'- (slot-value 'data))))
    (format t "Summary for ~a~%"
    (if (slot-value 'title) (slot-value 'title) "a paired sample"))
    (format t "The sample size is ~d~%"
    (length diffs))
    (format t "The mean difference is ~,4g~%"
    (mean diffs))
    (format t "The standard deviation of differences is ~,4g~%"
    (standard-deviation diffs))))

(defmeth paired-proto :display ()
  (if (slot-value 'title) (princ (slot-value 'title)))
  (let ((data (slot-value 'data))
    (print (reduce #'- data))
    (format t "The sample size is ~d~%" (length data))
    (terpri)))

(defmeth paired-proto :boxplot
  (boxplot (reduce #'- (slot-value 'data))))
```

4. (a) (load "/usr/people59/canty/teach/epfl/lispstat/plotmenu")

```
(def change-item1
     (send graph-item-proto :new "Unselected Symbol" plot1
   :change-symbol1))
(defmeth plot1 :change-symbol1 ()
   (let ((i (choose-item-dialog
       "New Plot Symbol"
       '("Dot" "Dot1" "Dot2" "Dot3" "Dot4"
         "Disk" "Diamond" "Cross" "Square"
         "Wedge1" "Wedge2" "X"))))
      (send self :point-symbol
           (which (mapcar #'not
                                     (send self :point-selected
              (iseq (send self :num-points)))))
   (select '(dot dot1 dot2 dot3 dot4
     disk diamond cross square
                              wedge1 wedge2 x) i))
      (send self :redraw)))
(send (send plot1 :menu) :append-items change-item1)
```

(b)

```
(def change-vars-item
     (send graph-item-proto :new "Change Variables" plot1
   :change-vars))
(defmeth plot1 :change-vars ()
  (let ((xvar (choose-item-dialog
         "Select New X variable"
 (list "Air" "Temp" "Conc" "Loss")
 :initial (first (send self :current-variables))))
(yvar (choose-item-dialog
         "Select New Y variable"
 (list "Air" "Temp" "Conc" "Loss")
 :initial (second (send self :current-variables)))))
    (send self :current-variables xvar yvar)
    (send self :adjust-to-data)))

(send (send plot1 :menu) :append-items change-vars-item)
```