Notes on the Computation for
“Partial Kelly Strategies and Expected Utility: Small Edge Asymptotics”
J.B. Kadane

I used the package R, available (free) at www.r-project.org.

(a) Table 2.

The computations for Table 2 use (10) in the paper. However, (10) is predicated on bets on $A$ and on $\bar{A}$, while the Blackjack problem is cast in terms of bets on $A$ and on cash. Hence, if $\ell$ is the amount found using (10), the amount bet in Blackjack is $\ell - (1 - \ell) = 2\ell - 1$.

Comments on the code are designated by lines starting #.

```
lam = c(0.8, 0.6, 0.5, 0.4, 0.2)
# Thus lam is a vector of length 5. R works readily with vectors
q = 0.51
x = 0.5
ell_1 = q * lam + x * (1 - lam)
y_1 = 2 * ell_1 - 1
# ell_1 is a vector of length 5 giving the partial-Kelly numbers.
# y_1 transforms them as remarked above.
# Now using (10), and noting that $x/\bar{x} = 1$,
# $\ell = (q/\bar{q})^{-1/\theta} = (q/\bar{q})^{1/\theta} = \left(\frac{q}{\bar{q}}\right)\lambda = w$.
# so $\ell = w/(1 + w)$
w = (q/(1 - q)) * lam
# ** indicates “to the power”
ell_2 = w/(1 + w)
y_2 = 2 * ell_2 - 1
# y_2 is the approximating value
```

(b) Table 4

1
The computations here are more complicated, because there are six ways to invest (5 horses and cash), and there are $2^5 = 32$ possible outcomes to take into account (each of the 5 horses wins or does not).

To account for the first issue, it is necessary to have software that maximizes a (non-linear) function subject to linear inequality constraints. The software I used is called “constrOptim”. Details on how to use it can be found by typing

`help(constrOptim, package=stats)` in R.

The natural constraints in this problem are convex coefficients,

$$p_i \geq 0, \ i = 1, \ldots, 6, \ \sum_{i=1}^{6} p_i = 1,$$

(1)

where $p_i$($i = 1, \ldots, 5$) is the proportion of fortune to bet on each horse, and $p_6$ is the proportion to leave in cash. The program constrOptim wants constraints in the form

$$(u)p - c \geq 0$$

(2)

where $u$ is the matrix, $c$ and 0 are vectors, and $p$ is the vector of parameters to be optimized. To accomplish this, first one of the $p_i$’s must be removed from the optimization space. Arbitrarily I choose $p_6$, the proportion kept in cash, so the optimization is over $p_1, \ldots, p_5$, subject to

$$p_i \geq 0 \ \ i = 1, \ldots, 5$$

and

$$1 - \sum_{i=1}^{5} p_i \geq 0.$$
This is cast in the framework of (2) by defining

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \\
-1 & -1 & -1 & -1 & -1
\end{bmatrix}
\]

and

\[
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
-1
\end{bmatrix}
\]

To deal with the second issue, it is useful to expand the integers from 0 to 31 in binary. The following function does that:

\[
\text{bi = function (n)}
\{
\text{m = n}
\text{z = NULL}
\text{for (i in 4:0)}
\{
\text{x = floor (m/(2**i))}
\text{m = m-((2**i)*x)}
\text{z = c(z,x)}
\text{next}
\}
\text{return (z)}
\}
\]

To explain what is happening here, the first line declares that \text{bi} is a function of a
single input, “n”. The second two lines declare starting values for m and z, respectively. The notation 4:0 means the set \{4,3,2,1,0\} in that order. So the code in this line is starting a loop of length 5, closed off by the line marked ‘next’. The floor function is the largest integer smaller than or equal to its argument. Thus in the first iteration of the for loop, if n is less than 16, the value of x is zero, while for n’s 16 or more, the value is 1. The next line subtracts from m, the value \(x \times 2^i\), and x is appended to z. The new number m is now between 0 and 15, and is compared to \(2^3 = 8\), etc. Thus the output z is a vector of 0’s and 1’s of length 5, showing the binary expansion of n. You can check this out by testing it. For example,

\[ b_i(24) = 11000 = 2^4 + 2^3 = 16 + 8 = 24. \]

The next step is to construct the utility function. I’ll start with log utility. Since constrOptim minimizes functions, I’ll need to insert a negative sign. First I need to declare two vectors:

\[ q = c(0.57, 0.38, 0.285, 0.228, 0.190) \]
\[ x = c(0.5, 1/3, 0.25, 0.20, 1/6). \]

Now I write a function to evaluate -log utility:

```r
lutil = function(p){
    co = NULL
    fi = NULL
    for (i in 0:31){
        cof = prod(q**bi(i))*prod((1-q)**(1-b(i)))
        co = c(co,cof)
        f = -log((1-sum(p))+sum(bi(i)*(p/x)))
        fi = c(fi,f)
    }
    return(sum(co*fi))
}
```

4
To explain, \( \mathbf{co} \) is built as a vector of length 32 giving the subjective probabilities of each of the 32 possible outcomes of the 5 races. The vector \( \mathbf{fi} \) is the \(-\log (\text{fortune})\) that results from each of those 32 outcomes, where \( 1-\sum(p) \) is the cash that was not risked, \( p/x \) is the vector of length 5 giving the number of tickets bought on each of the 5 horses, so the \( \sum(bi(i) \times (p/x)) \) is the amount won on these various bets. Then \( \mathbf{co} \mathbf{fi} \) is a vector of length 32, the element by element product of \( \mathbf{co} \) and \( \mathbf{fi} \), whose sum is the desired quantity.

To develop a program for the approximation, the log-utility program can be used, substituting (3) for \( f \). At this point, it is important to distinguish the cases \( \theta > 1 \) and \( \theta < 1 \). When \( \theta > 1 \), in (3), we can omit the factor \( 1/(\theta - 1) \), because it is positive and constant with respect to \( p \). When \( \theta < 1 \), this factor is negative, an additional factor of -1 must be used. Multiplying by sign \( (1-\lambda) \) allows both cases to be considered simultaneously. Also the “1-” in the numerator can be omitted, because the 1 will be summed over all the 32 probabilities of outcomes, and hence will have the (irrelevant) value 1 in the expected utility. Also, the minus sign should be omitted because constrOptim minimizes. Hence we are left with the form \( f^{1-\theta} \). Thus replacing

\[
\begin{align*}
  f &= -\log((1 - \sum(p) + \sum(b(i) \times (p/x))))
\end{align*}
\]

with

\[
\begin{align*}
  f &= (\text{sign}(1 - \lambda)) \times (1 - \sum(p) + \sum(b(i) \times (p/x)))^{(1 - 1/\lambda)},
\end{align*}
\]

provided a value for \( \lambda \) is specified, will deliver a value for \( \text{util} \). Written out, the function is

\[
\begin{align*}
  \lambda &= \text{[a value from 2, 1.5, .8, .6, .5, .4, .2]} \\
  \text{util} &= \text{function(p)} \\
  \mathbf{co} &= \text{NULL} \\
  \mathbf{fi} &= \text{NULL}
\end{align*}
\]
for (i in 0:31)
  cof = prod(q**bi(i))*prod((1-q)**(1-b(i)))
  co = c(co,cof)
  f = (sign(1-lambda))*(((1-sum(p) + sum(b(i)*(p/x))))**(1-1/lambda)
  fi = c(fi,f)
return (sum(co*fi))

The other input required by constrOptim is starting value (which also tells the program how many variables are to be optimized with respect to.) The starting value cannot be on the boundary of the space permitted by the constraints. I use

\[ p = c(1/6, 1/6, 1/6, 1/6, 1/6). \]

Then the call to the function looks like

constrOptim(p,lutil,NULL, u,ci).

[The “NULL” is a place for a gradient function, which I didn’t use.]

For the approximation, I used

\[ \lambda = 0.5 \]

# for example
constrOptim (p,util,NULL,u,ci).

The value for \( p_0 \), the proportion kept in cash, is found by subtraction from 1.