A simple options training model

LESTER INGBER

DRW Investments LLC, Chicago Mercantile Exchange Center

30 S Wacker Dr, Ste 1516, Chicago, IL 60606

and

Lester Ingber Research

PO Box 06440, Wacker Dr PO - Sears Tower, Chicago, IL 60606-0440

ingber@ingber.com ingber@alumni.caltech.edu

ABSTRACT

Options pricing can be based on sophisticated stochastic differential equation models. However, many traders, expert in their art of trading, develop their skills and intuitions based on loose analogies to such models and on games designed to tune their trading skills, not unlike the state of affairs in many disciplines. An analysis of one such game reveals some simple but relevant probabilistic insights into the nature of options trading often not discussed in most texts.

Keywords: options; volatility; optimization

1. INTRODUCTION

The pricing of financial options, even at the most rudimentary technical level [1], is relatively mathematically sophisticated for many practicing traders. While many options traders use numerical results output by computer-coded options models, their analytic and intuitive skills generally are honed by their active participation in trading.

This should come as no surprise, as similar circumstances/gaps exist between analysts and practitioners in many disciplines, e.g., as evidenced by relatively sophisticated studies in finance [2], neuroscience [3,4], combat analysis [5], etc., which are not studied by many practitioners in these fields. This does not imply that many practitioners have no interest or use for such analyses, quite the contrary,

but they recognize that the sophistication of their discipline often relies heavily on teamwork among several to many kinds of experts.

At the Chicago Mercantile Exchange (CME), groups of traders, new as well as experienced, tune their floor trading skills, e.g., requiring aggressive rapid decision-making on bids and asks, hand signals, etc., by practicing trading games when they are off the floor. I was approached by such a group of traders, who wondered if there was any way I could determine whether their rather consistent results of a trading game could be determined as "fair" pricing by any analysis.

In the process of a quick examination of their trading game, I realized that my analysis could serve to introduce the technical layperson to some sense of how the activity of trading determines a fair price for options.

Section 2 is an outline of one variation of the trading game. Section 3 gives the analysis, stressing simple probability concepts. Appendices A and B give the C-code used to calculate the fair price of this options game. Section 4 is the conclusion.

2. THE GAME

A number N of cardboard cards are prepared, numbered from 1 to N on one side. Each of M participants draw one card. Typically, N is about 20−40 and M is 5−7. Some variations include placing a few cards either face up on down on the table.

Each trader must estimate the sum of the numbers on all cards held by the other traders (plus the numbers of face up cards on the table, etc.) Based on the sum of all possible numbers on all held cards, C_s, a set of trading positions is assumed. For example, in the spread game analyzed here, the maximum and minimum sum of card values C is given by

$$C_{\rm min} = \sum_{m=1}^{M} m$$

$$C_{\rm max} = \sum_{m=N-M+1}^{N} m \tag{1}$$

with a center value C_0 given by their average. (If $C_{\rm max} + C_{mim}$ is odd, 2 points span C_0.) The value of the option spread, a "straddle," is determined by the sum of the cards,

$$V_s = |C_s - C_0| \ . \tag{2}$$

Clearly, all position values are simply linearly increasing functions from the center zero position (or pair of positions of N is even, etc.).

After some time of declaring bids and asks, e.g., 15 minutes, a full accounting of positions is rendered.

Since the deal of cards introduces a statistical element into the game, some natural questions arise. For example, what is the average expected value of the spread position? Or, for a range of N and M, what is the largest value of V to be expected?

3. THE ANALYSIS

3.1. Expected Values

At the outset, I was able to show the traders that the expectation of the fair price of the spread, V, could be broken up into two parts, where the expectation is developed as the sum of products of probabilities, p_s, a given spread position, s, exists in the cards, times the values V_s of these positions,

$$V = \sum_s p_s V_s \ . \tag{3}$$

As mentioned previously, in this particular game, the positions V_s are simple linear functions. The problem then reduces to enumeration of the probabilities p_s of occurrences of states of the "underlying market" defined by the cards.

At first glance, it would seem wise to simply write down some combinatoric expressions to define this game. However, being armed with the knowledge of the precocious nature of traders, who will quickly evolve new games and new questions, and given the potentially large combinatoric expressions that could develop, I decided that a simulation, capable of inserting new constraints, etc., would be a better approach.

3.2. The Simulation

The simulation consists of many iterations of shuffling and dealing cards to the traders, adding up probabilities of various states, etc. Appendix A gives the C code for this simulation.

The code also permits the user to specify a number of cards up or down. In this particular code, cards up are included in the summation of cards, e.g., as if they are extra players. Cards down are not included in the sum. In either case, cards taken from the deck are not shuffled in the many iterations. To keep the code self-contained, values of the cards placed up or down can be directly coded into search.c.

As expected, for N large, e.g., even greater than 10, combinatorics quickly become very well approximated by a Gaussian distribution [6]. For example, for N = 30, M = 7, and 1,000,000 iterations, the actual computed moments about the mean of the distribution of the numbers held by the traders are given by

$$< 1 >= 1$$

$$< C >= 108.51$$

$$< (C - < C >)^2 >= 415.20$$

$$< (C - < C >)^3 >= -1.62$$

$$< (C - < C >)^4 >= 486038$$

$$\text{skew} = \frac{< (C - < C >)^3 >}{< (C - < C >)^2 >^{3/2}} = -0.00019$$

$$\text{kurtosis} = \frac{< (C - < C >)^4 >}{< (C - < C >)^2 >^2} = 2.82 \qquad (4)$$

The relatively small value of the skew and the relatively small deviation of the kurtosis from 3 shows that indeed a Gaussian distribution is a rather good approximation.

3.3. Some Results

For N = 30, M = 7, and 1,000,000 iterations, the most likely value of the spread given by the simulation is 16.39. Fig. 1 illustrates the distribution and the spread versus the card values. The traders have played this game for some years, and they have noted that their games regularly results in a best value of about 19. To get some perspective on the influence of the number of iterations, Fig. 2 gives the same game but with only 10,000 iterations.

Figure 1

Figure 2

While in accord with their experience, a bit surprising to the traders were the results for placing some cards up. There was not much difference noted. Since cards up are counted, the best comparison to the above example is for a case such as $N = 30$, $M = 4$, cards up = 3, and 1,000,000 iterations. Here the average value of 16.99 is close to the above example of 16.39. The additional information from the cards up reduces the volatility,

$$\text{volatility} = \sqrt{< (C - < C >)^2 > \frac{(\max - \min + 1)}{(\max - \min)}} \tag{5}$$

from 20.44 to 16.23.

For a cards down case, $N = 30$, $M = 7$, cards down = 3, and 1,000,000 iterations, an expected value of 16.37 and volatility of 20.01 was obtained.

The value of the option is about the same in all 3 cases (and not much different than runs using 10,000 iterations or less). There are several aspects and trade-offs to consider, e.g., relative peaking of the volatilities, different values of the sums, shifts of the peaks of the distributions, etc. For instance, the mean shifts for both cards up and down, but the sharper peak for cards up likely is more sensitive to picking up larger values of the spread as it moves off center from the case of no cards up or down.

3.4. Optimization Study

To answer some questions on maximum values of positions, etc., the spread.c code given in Appendix A was used as a (negative) cost function to be processed by Adaptive Simulated Annealing (ASA). ASA is a powerful global optimizer which can be obtained at no charge from http://www.ingber.com/ or http://www.alumni.caltech.edu/~ingber/ under WWW. ASA can optimize mixed integer and continuous variables under complex constraints.

The differences ("diffs") between the code available from the above InterNet archive and that used for this project is given in Appendix B. The diffs may be used conveniently as a patch file to easily generate the full optimization code for this project.

Since the statistics of this game are discrete, which becomes more pronounced at smaller values of all variables considered, it should not be surprising that a full global optimization is required to find the value of M, for a given value of N, that maximizes the value of the option.

For example, N was permitted to range from 2 to 100, and M from 2 to 100 subject to M less than N. There were no cards up or down, and each cost function used 10,000 iterations. (The traders wanted to see if indeed the largest value was for N = 100.) The value N = 100 gives the largest value since the positions range to higher values for higher N. Even optimizing the value of the option scaled by $1/N$ gives N = 100 as the largest value of the option.

A bit surprising to the traders was that the maximum value of the option for N = 100 was not M = 50 (based on binomial distribution combinatorics), but rather 38, yielding a value of 111.44 with volatility 139.51. This is a reflection of the importance of the discrete statistics at lower values of M.

3.5. Application to Real Trading

Real-world options trading, at least to a first order approximation is based on taking options positions on an underlying market. where the market is described by a lognormal distribution. The standard partial-differential equation used to formulate most variants of Black-Scholes (BS) models describing the market value of an option, V, is

$$\frac{\partial V}{\partial t} + \frac{1}{2}\,\sigma^2 S^2\,\frac{\partial^2 V}{\partial S^2} + rS\,\frac{\partial V}{\partial S} - rV = 0\ , \tag{6}$$

where S is the asset price, and σ is the standard deviation, or volatility of S, and r is the short-term interest rate. The solution depends on boundary conditions, subject to a number of interpretations, some requiring minor transformations of the basic BS equation or its solution. The basic equation can apply to a number of one-dimensional models of interpretations of prices given to V, e.g., puts or calls, and to S, e.g., stocks or futures, dividends, etc. For instance, if V is set to c, a call on an European option with exercise price E with maturity at T, the solution is

$$c(S,t) = SN(d_1) - Ee^{-r(T-t)}N(d_2)\ ,$$

$$d_1 = \frac{\ln(S/E) + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma(T-t)^{1/2}} ,$$

$$d_2 = \frac{\ln(S/E) + (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma(T-t)^{1/2}} . \tag{7}$$

Improvements to this approximation are the meat of many papers and investigations [2,7-9].

If a lognormal transformation is made to the market S described just above, with respect to some reference state S_0

$$Z = \ln(C/C_0) \tag{8}$$

then the transformed partial differential equation is Gaussian with respect to Z. This suggests how the above game/model, with an underlying market C described by values of held cards, might be recast into a more realistic market.

For example, consider the Eurodollar market. Eurodollars are fixed-rate time deposits held primarily by overseas banks, but denominated in US dollars. They are not subject to US banking regulations and therefore tend to have a tighter bid-ask spread than deposits held in the United States [10]. The three-month Eurodollar futures contract is one of the most actively traded futures markets in the world. The contract is quoted as an index where the yield y is equal to the Eurodollar price E subtracted from 100,

$$y_t = 100 - E_t . \tag{9}$$

This yield is equal to the fixed rate of interest paid by Eurodollar time deposits upon maturity and is expressed as an annualized interest rate based on a 360-day year. The Eurodollar futures are cash settled based on the 90-day London Interbank Offer Rate (LIBOR). A “notional” principal amount of $1 million, is used to determine the change in the total interest payable on a hypothetical underlying time deposit, but is never actually paid or received [10].

If we consider

$$Z_t = \ln \frac{y_t}{y_{t-1}} \approx \frac{y_t - y_{t-1}}{y_{t-1}} , \tag{10}$$

and consider the values of the sums of cards in the above game as representing percentage deviations of Eurodollar yields, then there is a somewhat realistic mapping of this game onto a model of a real market.

(The spread values must be similarly transformed.)

4. CONCLUSION

Options pricing is regularly based on relatively sophisticated stochastic differential equations. However, most textbooks to not develop the basic intuitions behind these equations as arising from rather simple probabilistic concepts, e.g., expected value as a sum over probabilities of states of the underlying market times the values of the options positions at these states. This paper presents an options model with such an interpretation, based on an options training game.

With the inevitable rapid decline of open-outcry floor trading. as electronic trading is rapidly gaining popularity among investors, all traders must become more cognizant of the value of sophisticated analyses to their bottom line net income. Such analyses and training games as presented here can help to develop an awareness of such analyses.

It is my contention that similar circumstances exist in many disciplines, ranging from finance, to neuroscience, to combat analyses, etc., disciplines which are blends of Art and Science. Many people in these disciplines are in danger of extinction if they do not increase their training in modern analyses and technologies.

APPENDIX A: SPREAD C-CODE

The spread code has two parts, spread.c, the main code, and spread.h, an "include" file. The "SPREAD_ASA" statements are used to permit the spread code to be run as is, or included in the optimizing code Adaptive Simulated Annealing (ASA) [11].

A.1 spread.c

```
/* SPREAD_ASA is set to TRUE=1 in ASA when patch.GAMES is applied */
#ifndef SPREAD_ASA
#define SPREAD_ASA 0
#endif

#if SPREAD_ASA
#else /* SPREAD_ASA */
/* spread cards peop up_cards dn_cards iter */

#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main (int argc, char **argv);

#include "spread.h"

int
main (int argc, char **argv)
{
#endif /* SPREAD_ASA */
  int m, n, i, nn, n_up, n_dn;
  int cards, peop;
  int up_cards, dn_cards;
  int min, max;
  int *deck, *up_deck, *dn_deck, *tmp_deck;
  long int r, iter;
  long int norm, sum;
  double av, vol, *prob, payoff, value, mid_value, valpercard;
  double xx, y, m0, m1, m2, m3, m4, skew, kurtosis;
  char spread_out[80];
  FILE *ptr_out;

#if SPREAD_ASA
  cards = x[0];
  peop = x[1];
  iter = 10000;
  up_cards = 0;
  dn_cards = 0;
#else /* SPREAD_ASA */
  cards = atoi (argv[1]);
```

```
  peop = atoi (argv[2]);
  up_cards = atoi (argv[3]);
  dn_cards = atoi (argv[4]);
  iter = atol (argv[5]);
#endif /* SPREAD_ASA */

  if (peop > cards - up_cards - dn_cards)
    {
#if SPREAD_ASA           /* SPREAD_ASA */
      *cost_flag = FALSE;
      return (0);
#else /* SPREAD_ASA */
      exit (-6);
#endif /* SPREAD_ASA */
    }

  if ((deck = (int *) calloc (cards, sizeof (int))) == NULL)
      exit (9);
  if ((up_deck = (int *) calloc (cards, sizeof (int))) == NULL)
      exit (9);
  if ((dn_deck = (int *) calloc (cards, sizeof (int))) == NULL)
      exit (9);
  if ((tmp_deck = (int *) calloc (cards, sizeof (int))) == NULL)
      exit (9);

  av = 0;            /* use when cards = peop */
  for (n = 0; n < cards; ++n)
    {
      deck[n] = n + 1;
      tmp_deck[n] = deck[n];
      up_deck[n] = dn_deck[n] = 0;
      av += (double) deck[n];
    }
  av /= (double) cards;
  vol = 0;

  /* calc midpoint of spread */
  min = 0;
  for (m = 0; m < peop + up_cards; ++m)
    {
      min += deck[m];
    }
  max = 0;
  for (m = cards; m > cards - peop - up_cards; --m)
    {
      max += deck[m - 1];
    }
  mid_value = (double) (max + min) / 2.0;

  if ((prob = (double *) calloc (max + 1, sizeof (double))) == NULL)
      exit (9);

  for (n = 0; n < max + 1; ++n)
    {
```

```
      prob[n] = 0;
    }

  /* select possible up and dn cards */
  up_deck[0] = 9;
  up_deck[1] = 20;
  up_deck[2] = 2;

  dn_deck[0] = 9;
  dn_deck[1] = 20;
  dn_deck[2] = 2;

  if (up_cards == 0)
    {
      ;
    }
  else
    {
      for (n_up = 0; n_up < up_cards; ++n_up)
    {
      nn = 0;
      for (n = 0; n < cards; ++n)
        {
          if (deck[n] == up_deck[n_up])
        {
          ;
        }
          else
        {
          tmp_deck[nn] = deck[n];
          ++nn;
        }
        }
      for (n = 0; n < cards; ++n)
        {
          deck[n] = tmp_deck[n];
        }
      --cards;
    }
    }

  if (dn_cards == 0)
    {
      ;
    }
  else
    {
      for (n_dn = 0; n_dn < dn_cards; ++n_dn)
    {
      nn = 0;
      for (n = 0; n < cards; ++n)
        {
          if (deck[n] == dn_deck[n_dn])
        {
```

```
          ;
        }
          else
        {
          tmp_deck[nn] = deck[n];
          ++nn;
        }
        }
      --cards;
      for (n = 0; n < cards; ++n)
        {
          deck[n] = tmp_deck[n];
        }
    }
    }

  m0 = m1 = m2 = m3 = m4 = skew = kurtosis = 0;

  if (max > min)
    {
      /* warm up random() */
      for (i = 0; i < 100; ++i)
    {
      shuffle (deck, cards);
    }

      norm = 0;
      for (r = 0; r < iter; ++r)
    {
      shuffle (deck, cards);
      sum = 0;
      for (m = 0; m < peop; ++m)
        {
          sum += (long int) deck[m];
        }
      /* dn_cards not included in sum */
      if (up_cards > 0)
        {
          for (n = 0; n < up_cards; ++n)
        {
          sum += (long int) up_deck[n];
        }
        }
      prob[sum] += 1.0;
      ++norm;
    }

      av = 0;
      vol = 0;
      for (m = min; m <= max; ++m)
    {
      prob[m] /= (double) norm;
      av += ((double) m) * (prob[m]);
      vol += ((double) m) * ((double) m) * (prob[m]);
```

```
    }

      vol -= av * av;
      vol = sqrt (((double) (max - min + 1) / (double) (max - min)) * vol);

      for (m = min; m <= max; ++m)
    {
      xx = prob[m];
      y = (double) m - av;
      m0 += xx;
      m1 += xx * y;
      m2 += xx * y * y;
      m3 += xx * y * y * y;
      m4 += xx * y * y * y * y;
    }
      skew = m3 / pow (m2, 1.5);
      kurtosis = m4 / (m2 * m2);
    }

  sprintf (spread_out, "%s_%d_%d_%d_%d_%ld", "spread",
       cards + up_cards + dn_cards, peop, up_cards, dn_cards, iter);

  if ((ptr_out = fopen (spread_out, "w")) == NULL)
    exit (-6);

  fprintf (ptr_out,
     "cards = %d\tpeop = %d\tup_cards = %d\tdn_cards= %d\titer = %ld\n",
       cards + up_cards + dn_cards, peop, up_cards, dn_cards, iter);

  for (n = 0; n < up_cards; ++n)
    {
      fprintf (ptr_out, "up_deck[%d] = %d\n", n, up_deck[n]);
    }
  for (n = 0; n < dn_cards; ++n)
    {
      fprintf (ptr_out, "dn_deck[%d] = %d\n", n, dn_deck[n]);
    }

  fprintf (ptr_out, "cards\tprob\tpayoff\tprod\n");
  value = 0;
  for (m = min; m <= max; ++m)
    {
      payoff = fabs ((double) m - mid_value);
      value += prob[m] * payoff;
      fprintf (ptr_out, "%d\t%g\t%g\t%g\n",
           m, prob[m], payoff, prob[m] * payoff);
    }
  valpercard = value / (double) cards;

  fprintf (ptr_out, "m0 = %g\tm1 = %g\tm2 = %g\tm3 = %g\tm4 = %g\n",
       m0, m1, m2, m3, m4);
  fprintf (ptr_out, "skew = %g\tkurtosis = %g\n",
       skew, kurtosis);
```

```
  fprintf (ptr_out, "Average Sum Cards = %g\tVol = %g\n", av, vol);

  fprintf (ptr_out, "Value = %g\tValue/Card = %g\n",
       value, valpercard);

  fclose (ptr_out);

  free (prob);
  free (up_deck);
  free (dn_deck);
  free (deck);

#if SPREAD_ASA
#if 0
  return (-valpercard);
#else
  return (-value);
#endif
#else /* SPREAD_ASA */
  exit (0);
  /* NOTREACHED */
}
#endif /* SPREAD_ASA */
```

A.2 spread.h

```
void shuffle (int *deck, int cards);
void swap (int *p, int *q);

void
shuffle (int *deck, int cards)
{
  int n, p;

  for (n = 0; n < cards; ++n)
    {
      p = (int) (random () % (long int) cards);
      swap (&(deck[n]), &(deck[p]));
    }
}

void
swap (int *p, int *q)
{
  int t;

  t = *p;
  *p = *q;
  *q = t;
}
```

APPENDIX B: PATCH TO OPTIMIZE SPREAD C-CODE

The differences ("diffs") between the code available from http://www.ingber.com/ and that used for this project may be used conveniently as a patch file to easily generate the full optimization code for this project. The code below was prepared from ASA version 17.8.

B.1 patch

```
diff -rc ASA/Makefile GAMES/Makefile
*** ASA/Makefile Sun Jul 12 09:49:45 1998
--- GAMES/Makefile   Mon Aug 24 09:34:29 1998
***************
*** 49,55 ****
  ## lines; if the latter, be sure each line to be continued ends in a "\"
  ## (backslash).

! DEFINE_OPTIONS = -DASA_TEST=TRUE # -DMY_TEMPLATE=TRUE

  ## This will run the ASA problem.
  #DEFINE_OPTIONS = -DASA_TEST=TRUE
--- 49,55 ----
  ## lines; if the latter, be sure each line to be continued ends in a "\"
  ## (backslash).

! DEFINE_OPTIONS = # -DMY_TEMPLATE=TRUE

  ## This will run the ASA problem.
  #DEFINE_OPTIONS = -DASA_TEST=TRUE
***************
*** 411,417 ****
  ## The gnu C compiler is the default.
  #CC = g++
  CC = gcc
! CDEBUGFLAGS = -g -O -Wall # MY_TEMPLATE_flags
  ## If you wish to include some profile statistics
  #CDEBUGFLAGS = -g -O -pg -Wall
  ##
--- 411,417 ----
  ## The gnu C compiler is the default.
  #CC = g++
  CC = gcc
! CDEBUGFLAGS = -g -O2 -Wall # MY_TEMPLATE_flags
  ## If you wish to include some profile statistics
  #CDEBUGFLAGS = -g -O -pg -Wall
  ##
***************
*** 473,478 ****
--- 473,481 ----
  #compile: $(USEROBJS) $(ASAOBJS)
  # @$(CC) $(LDFLAGS) -o asa_run $(USEROBJS) $(ASAOBJS) /usr/local/lib/leak.o -lm
```

```
+ spread:
+   gcc -g -O2 -Wall -o spread spread.c -lm
+
  ## If COST_FILE is set to FALSE, user_cst.h may be deleted.  If the
  ## name user_cst.h is changed, then this might be changed here as well.
  $(USEROBJS): user.h user_cst.h # MY_TEMPLATE_user_incl
***************
*** 491,501 ****
  # profile

  clean:
!   rm -f *\.o asa_run # MY_TEMPLATE_clean core gmon.out

  realclean:
    rm -f *\.o asa_run user_out* asa_out* core asa_save* asa_rcur asa_sfop \
!   asa_[A-D]_[a-d] # MY_TEMPLATE_realclean gmon.out

  docclean:
    rm -f ASA-README ASA-README.ps
--- 494,504 ----
  # profile

  clean:
!   rm -f *\.o asa_run spread spread_* # MY_TEMPLATE_clean core gmon.out

  realclean:
    rm -f *\.o asa_run user_out* asa_out* core asa_save* asa_rcur asa_sfop \
!   asa_[A-D]_[a-d] spread spread_* # MY_TEMPLATE_realclean gmon.out

  docclean:
    rm -f ASA-README ASA-README.ps
diff -rc ASA/asa_opt GAMES/asa_opt
*** ASA/asa_opt  Sun Jul 12 09:49:47 1998
--- GAMES/asa_opt    Mon Aug 24 09:23:49 1998
***************
*** 1,5 ****
! Limit_Acceptances[10000][ASA_TEST:1000]         1000
! Limit_Generated[99999]                  99999
  Limit_Invalid_Generated_States[1000]            1000
  Accepted_To_Generated_Ratio[1.0E-6][ASA_TEST:1.0E-4]1.0E-4

--- 1,5 ----
! Limit_Acceptances[10000][ASA_TEST:1000]         30
! Limit_Generated[99999]                  50
  Limit_Invalid_Generated_States[1000]            1000
  Accepted_To_Generated_Ratio[1.0E-6][ASA_TEST:1.0E-4]1.0E-4

***************
*** 15,24 ****
  Sequential_Parameters[-1]               -1
  Initial_Parameter_Temperature[1.0]          1.0

! Acceptance_Frequency_Modulus[100]           100
! Generated_Frequency_Modulus[10000]          10000
```

```
! Reanneal_Cost[1]                    1
! Reanneal_Parameters[TRUE=1]             1

  Delta_X[0.001]                      0.001
  User_Tangents[FALSE=0]                   0
--- 15,24 ----
  Sequential_Parameters[-1]                -1
  Initial_Parameter_Temperature[1.0]           1.0

! Acceptance_Frequency_Modulus[100]           10
! Generated_Frequency_Modulus[10000]          100
! Reanneal_Cost[1]                    0
! Reanneal_Parameters[TRUE=1]              0

  Delta_X[0.001]                      0.001
  User_Tangents[FALSE=0]                   0
***************
*** 26,38 ****

  ___Define_below_if_OPTIONS_FILE_DATA=TRUE

! number_parameters=*parameter_dimension          4

  Param#:Minimum:Maximum:InitialValue:Integer[1or2]orReal[-1or-2]
! 0 -10000  10000   999.0            -1
! 1 -10000  10000   -1007.0          -1
! 2 -10000  10000   1001.0           -1
! 3 -10000  10000   -903.0           -1

  ___Define_below_if_QUENCH_COST_and_OPTIONS_FILE_DATA=TRUE

--- 26,36 ----

  ___Define_below_if_OPTIONS_FILE_DATA=TRUE

! number_parameters=*parameter_dimension          2

  Param#:Minimum:Maximum:InitialValue:Integer[1or2]orReal[-1or-2]
! 0 2   30  5            2
! 1 2   30  10           2

  ___Define_below_if_QUENCH_COST_and_OPTIONS_FILE_DATA=TRUE

diff -rc ASA/asa_user.h GAMES/asa_user.h
*** ASA/asa_user.h   Sun Jul 12 09:49:47 1998
--- GAMES/asa_user.h Mon Aug 24 09:33:18 1998
***************
*** 35,40 ****
--- 35,44 ----
  #endif
  #if MY_TEMPLATE             /* MY_TEMPLATE_asa_user */
    /* you can add your own set of #define here */
+ #ifndef SPREAD_ASA
+ #define SPREAD_ASA TRUE
```

```
+ #endif
+
  #endif

  #ifndef ASA_TEMPLATE_LIB
diff -rc ASA/user_cst.h GAMES/user_cst.h
*** ASA/user_cst.h   Sun Jul 12 09:49:46 1998
--- GAMES/user_cst.h Mon Aug 24 09:33:19 1998
***************
*** 26,31 ****
--- 26,33 ----
      x[0], ..., x[*parameter_dimension-1]
      for your parameters and to return the value of your cost function.  */

+ #include "spread.h"
+
  #if HAVE_ANSI
  double
  cost_function (double *x,
***************
*** 64,69 ****
--- 66,72 ----
  {

    /* *** Insert the body of your cost function here. *** */
+ #include "spread.c"

  #if ASA_TEST
    double q_n, d_i, s_i, t_i, z_i, c_r;
```

REFERENCES

1. J.C. Hull, ***Options, Futures, and Other Derivatives, Third Edition***, Prentice Hall, Upper Saddle River, NJ, (1997).

2. L. Ingber and J.K. Wilson, Volatility of volatility of financial markets, ***Mathl. Computer Modelling*** , (to be published) (1998).

3. L. Ingber, Statistical mechanics of neocortical interactions: Applications of canonical momenta indicators to electroencephalography, ***Phys. Rev. E*** **55** (4), 4578-4593 (1997).

4. L. Ingber, Statistical mechanics of neocortical interactions: Training and testing canonical momenta indicators of EEG, ***Mathl. Computer Modelling*** **27** (3), 33-64 (1998).

5. M. Bowman and L. Ingber, Canonical momenta of nonlinear combat, in ***Proceedings of the 1997 Simulation Multi-Conference, 6-10 April 1997, Atlanta, GA***, Society for Computer Simulation, San Diego, CA, (1997).

6. J. Mathews and R.L. Walker, ***Mathematical Methods of Physics, 2nd ed.***, Benjamin, New York, NY, (1970).

7. L. Ederington and W. Guan, Is implied volatility an informationally efficient and effective predictor of future volatility?, U Oklahoma, Norman, OK, (1998).

8. G. Bakshi, C. Cao, and Z. Chen, Pricing and hedging long-term options, Pennsylvania State U, University Park, PA, (1998).

9. L. Ingber, Some Applications of Statistical Mechanics of Financial Markets, LIR-98-1-SASMFM, Lester Ingber Research, Chicago, IL, (1998).

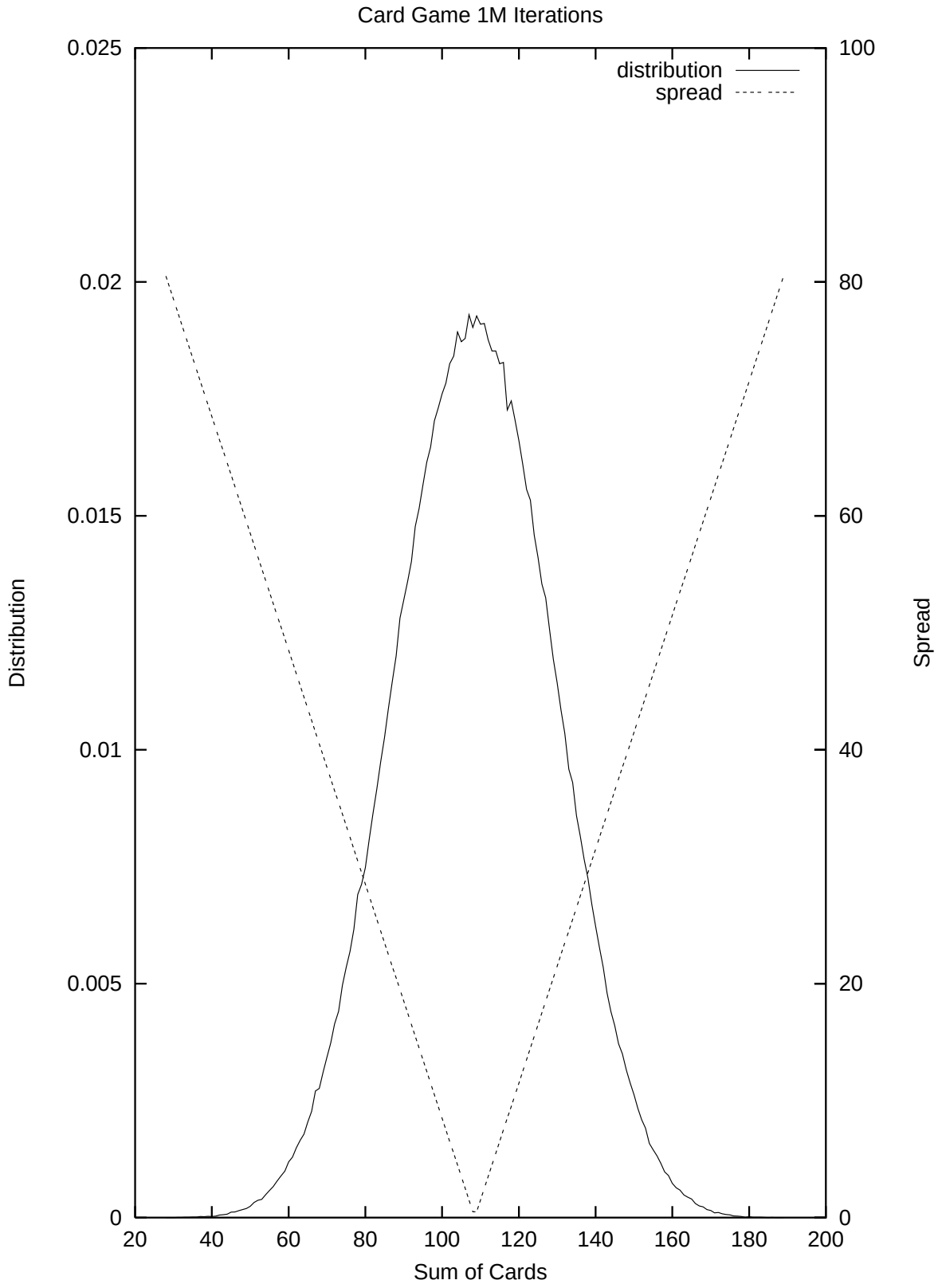
10. Federal Reserve Bank, ***Instruments of the Money Markets, Seventh Edition***, Federal Reserve Bank of Richmond, Richmond, VA, (1993).

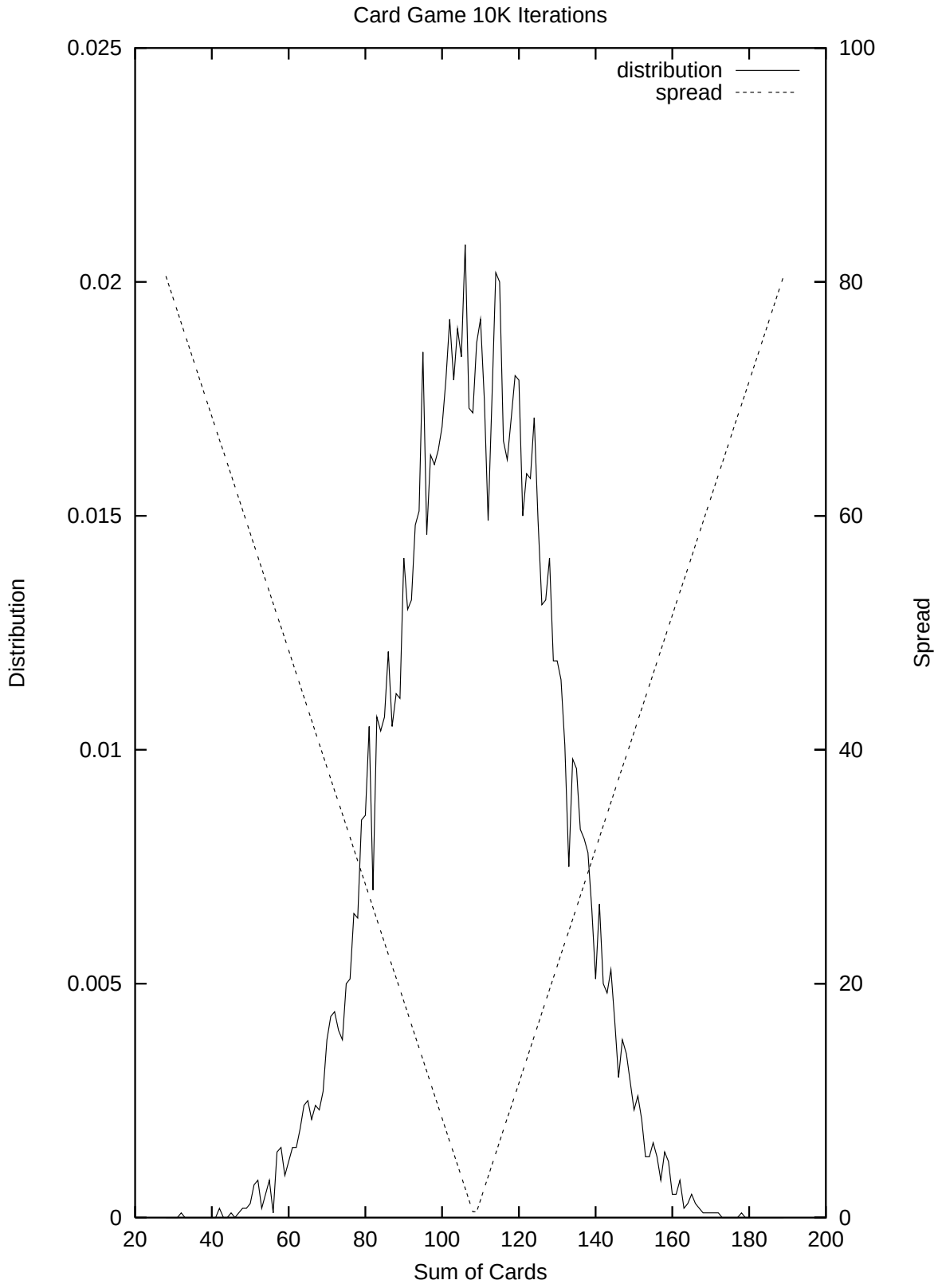
11. L. Ingber, Adaptive Simulated Annealing (ASA), Global optimization C-code, Lester Ingber Research, Chicago, IL, (1993).

FIGURE CAPTIONS

FIG. 1. The distribution of card values and their spread versus are given. There are 30 cards and 7 players. The number of iterations is 1,000,000.

FIG. 2. The distribution of card values and their spread versus are given. There are 30 cards and 7 players. The number of iterations is 10,000.


Card Game 1M Iterations
distribution
spread
Distribution
Spread
Sum of Cards
0.025
0.02
0.015
0.01
0.005
0
100
80
60
40
20
0
20
40
60
80
100
120
140
160
180
200


Card Game 10K Iterations
distribution
spread
Distribution
Spread
Sum of Cards
0.025
0.02
0.015
0.01
0.005
0
100
80
60
40
20
0
20
40
60
80
100
120
140
160
180
200