Asian Option Pricing with “GMP-(X)SC”

Knut Petras
TU Braunschweig

1. What is an Asian Option?

S_t: "stock" (or oil, or iron,...) price at time t
T: expiration time
X: strike price

pay-off: $$\phi(S) = \left(\frac{1}{T}\int_0^T S_t\,dt - X\right)_+$$

Purpose: "gambling" or "insurance against high price"

Black-Scholes' assumption:
S_t follows a geometric Brownian motion

2. Computation Methods I

Simulation (Monte-Carlo):
Expected error with n paths: $\sim n^{-1/2}$

PDE of the form

$$u_t + c_1 x^2 u_{xx} - (c_2 + c_3 x) u_x = 0,$$

$$\begin{aligned}
&u(T,x) = \frac{|x| - x}{2}, \\
&\lim_{x\to\infty} u(t,x) = 0, \\
&\lim_{x\to-\infty} x + u(t,x) = 0
\end{aligned}$$

Price:

$$C = S_0 u(0, \frac{X}{S_0})$$

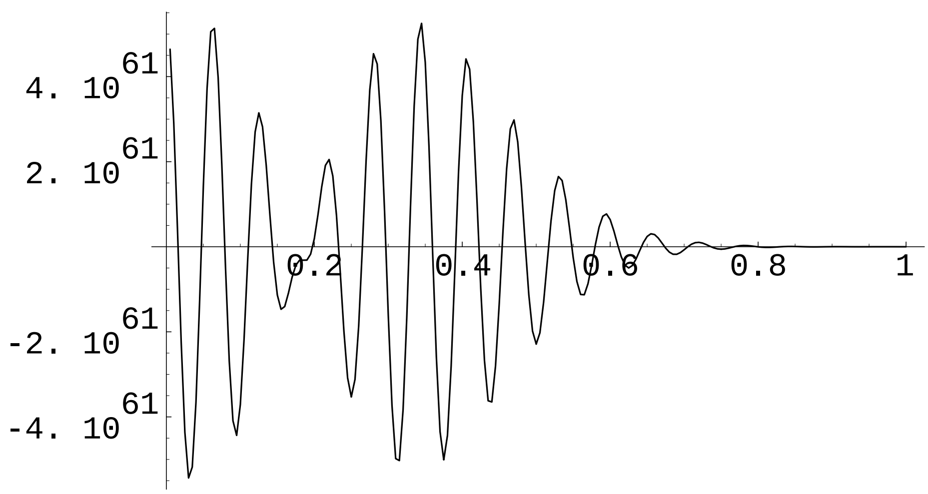
r	Strike	PDE_1	PDE_2	PDE_3	LB_1	LB_2	UB
0.05	95	6.147	6.826	7.157	7.174	7.178	7.183
		(2.07)	(7.95)	(29.4)	(0.03)	(2.95)	(0.61)
	100	3.009	2.577	2.621	2.713	2.716	2.722
		(2.07)	(7.95)	(29.4)	(0.03)	(2.95)	(0.61)
	105	1.289	0.641	0.439	0.334	0.337	0.343
		(2.07)	(7.95)	(29.4)	(0.03)	(2.95)	(0.61)
0.09	95	7.616	8.671	8.823	8.808	8.809	8.821
		(2.16)	(8.26)	(31.3)	(0.03)	(2.96)	(0.56)
	100	4.004	3.931	4.185	4.305	4.308	4.318
		(2.16)	(8.26)	(31.3)	(0.03)	(2.96)	(0.56)
	105	1.848	1.193	1.011	0.955	0.958	0.968
		(2.16)	(8.26)	(31.3)	(0.03)	(2.96)	(0.56)
0.15	95	10.026	11.223	11.090	11.094	11.094	11.114
		(2.47)	(9.28)	(45.0)	(0.02)	(2.96)	(0.59)
	100	5.829	6.447	6.777	6.790	6.794	6.810
		(2.47)	(9.28)	(45.0)	(0.02)	(2.96)	(0.59)
	105	2.990	2.582	2.639	2.741	2.744	2.761
		(2.47)	(9.28)	(45.0)	(0.02)	(2.96)	(0.59)

3. Schröder's Integral Representation

$$\mathsf{C} \;=\; \ldots + c \cdot \int_0^\infty H_\mu\left(\frac{\cosh x}{\sqrt{2q}}\right) \cdot e^{bx} \cdot$$
$$\cdot \sum_{|b| \in \{\nu, \nu+2\}} \Im\left(e^{i\pi b}\,\mathrm{erfc}\left(\frac{x + hb + i\pi}{\sqrt{2h}}\right)\right)\, dx$$

(with "some" derived quantities c, h, μ, q)

Integrand:



4. MATHEMATICA' s Numerical Integration

... from 0 to 5/2 (on 500 MHz-PC):

Input:

```
f=....
Timing[a=NIntegrate[f,{x,0,5/2}, MaxRecursion->20,
       WorkingPrecision->....]]
```

Output:

```
NIntegrate::ploss:
 Numerical integration stopping due to loss of precision.
 Achieved neither the requested PrecisionGoal nor
 AccuracyGoal; suspect one of the following: highly
 oscillatory integrand or the true value of the integral
 is 0. If your integrand is oscillatory try using the
 option Method->Oscillatory in NIntegrate.
```

digits	comp. time (s)	result
120	368.97	$0 * 10^{47}$
160	750.33	$0 * 10^{47}$
200	1325.12	$0 * 10^{47}$
220	540.34	$-3 * 10^{59}$
240	2250.49	$0 * 10^{40}$
260	2715.47	$0 * 10^{20}$
280	5625.93	$-1.249915... * 10^{12}$

5. Computation Methods II

Problem 1: unpredictable loss of many digits

Approach:

- "fast Multiple Precision" package (GMP)
- Subdivision; error control with interval techniques
  - a) interval computation and Automatic Differentiation or
  - b) complex interval arithmetic and Cauchy's integral formula

$$\frac{|f^{(n)}(x)|}{n!} \leq \frac{1}{2\pi} \int_{\partial\Omega} \frac{|f(z)|}{|z-x|^{n+1}} \, |dz|$$

Problem 2:

"Slow" multiple precision special functions

Approach:

- accelerate
- use derivatives

$$Q_{2\times m}[f] = \sum_{\mu=1}^{m} \frac{f^{(\mu-1)}(a) + (-1)^\mu f^{(\mu-1)}(b)}{\mu!} \times$$

$$\times (b-a)^\mu \frac{\binom{m}{\mu}}{\binom{2m}{\mu}}$$

Use derivatives; e.g.:

$$g(x) = H_\mu(c \cdot \cosh x), \qquad g_k := \frac{g^{(k)}(x)}{k!} =?$$

(Automatic Differentiation).

$$v = H'_\mu \circ w, \qquad w(x) = c \cdot \cosh x$$

$$g' = w' \cdot v \quad \Rightarrow \quad \boxed{g_k = \frac{1}{k} \sum_{i=1}^{k} i w_i v_{k-i}}$$

ODE for H_μ $\Rightarrow$

$$v' = 2[(ww')v - \mu w' g]$$

$$\Rightarrow \quad \boxed{v_k = \frac{2}{k} \sum_{i=1}^{k} [(ww')_{i-1} v_{k-i} - \mu i w_i g_{k-i}]}$$

costs: $O(k^2)$ multiplications

Comparison:

$H_\mu(x)$ with d correct digits costs $O(d)$ multiplications

6. What is GMP-(X)SC ?

GMP

Gnu Multiple Precision: C-library for arithmetic and $\sqrt{\ }$

Typical instructions:

```
mpf_t x,y,z;                /* declaration         */
mpf_init2(x,256);           /* memory allocation   */
mpf_init2(y,256);
mpf_init2(z,256);
mpf_set_si(x,3);            /* assignment x=3      */
mpf_set_d(y,0.25);          /* assignment y=0.25   */
mpf_mul(z,x,y);             /* z=x*y               */
mpf_out_str (NULL,10,30,z);
        /* output of z with 30 decimal digits */
mpf_clear(x); mpf_clear(x); mpf_clear(x);
        /* memory deallocation                */
```

GMP-SC program for the evaluation of one summand of the integrand

```
# include "gmp_sc.h"

void integrand(mpi_t y, mpi_t x, mpi_t h,
                        mpi_t b, mpi_t q, mpi_t mu) {
  mpi_t pi,h1,h2,h3;    /* intervals         */
  mpci_t ch1,ch2;       /* complex intervals */

  int dig=mpf_get_prec(y);

  mpi_init2(pi,dig); mpi_init2(h1,dig);
  mpi_init2(h2,dig); mpi_init2(h3,dig);
  mpci_init2(ch1,dig); mpci_init2(ch2,dig);

  mpi_pi(pi);

  /*** ch1 = erfc( ... ) : ***/
  mpi_mul(h1,b,h);
  mpi_add(h1,h1,x);
  mpci_set_ii(ch1,h1,pi);
  mpi_mul_2exp(h3,h,1);
  mpi_sqrt(h1,h3);
  mpci_div_i(ch1,ch1,h1);
  mpci_erfc(ch1,ch1);

  /*** ch2 = e^(i*pi*b) : ***/
  mpi_mul(h1,pi,b);
  mpi_set_si(h2,0);
  mpci_set_ii(ch2,h2,h1);
  mpci_exp(ch2,ch2);
```

```
  /*** h1 = Im (...) : ***/
  mpci_mul(ch1,ch1,ch2);
  mpi_set(h1,(*ch1).im);

  /** h2 = hermite(....) ***/
  mpi_sicoh(x,h3,h2);
  mpi_mul_2exp(h3,q,1);
  mpi_sqrt(h3,h3);
  mpi_div(h2,h2,h3);
  mpi_hermite(h2,h2,mu);   /** doesn't really exist **/

  /** final: ***/

  mpi_mul(h3,b,x);
  mpi_exp(h3,h3);
  mpi_mul(h1,h1,h2);
  mpi_mul(y,h1,h3);

  mpi_clear(pi); mpi_clear(h1); mpi_clear(h2);
  mpi_clear(h3); mpci_clear(ch1); mpci_clear(ch2);
}
```

GMP-XSC program for the evaluation of one summand of the integrand

```
# include "gmp_xsc.h"

void integrand(mpi y, mpi x, mpi h, mpi b, mpi mu) {
  int dig=get_prec(y);
  mpi pi(get_prec(y));
  pi=Pi(pi);
  y=hermite(cosh(x)/sqrt(2*q),b)*epx(b*x)
    *Im(exp(complexi(intval(0,0), pi*b))
    *erfc(complexi(x+b*h,pi)/sqrt(2*h)));
  del(pi);
}
```

Additional feature of GMP-SC for
fast enclosure of higher derivatives

Problems:

- Multiple Precision doesn't help much
- floating point ops: often overflow

Solution:

Numbers (+ arithmetic) with very large exponents

```
typedef struct {
  double num;
  int expo;} Large_d;
```

Intervals (+ arithmetic) with very large exponents

```
typedef struct {
  large_d lb;
  large_d ub;} Large_i;
```

7. Timing

Speed-up factors compared with PASCAL-XSC

Multiple precision numbers with 640 binary digits

function	SC	XSC	function	SC	XSC
$+$	10	5.5	$\sqrt{\ }$	16.9	16.8
$-$	10	6.1	exp	10.3	10.3
$*$	17.3	8.5	ln	40.7	40.7
/	10.3	9.6	sin	17	17
/int	27	19.3	arctan	38.2	38.2

Multiple precision intervals with 640 binary digits

function	SC	XSC	function	SC	XSC
$+$	3.5	2.8	$\sqrt{\ }$	16.1	16.1
-	3.5	2.8	exp	9.5	9.5
$*$	2.8	2.4	ln	39.7	39.7
/	6.5	6.4	sin	24.5	24.5
/int	11.4	9.5	arctan	64.7	64.7

MATHEMATICA

digits	comp. time (s)	result
120	368.97	$0 * 10^{47}$
160	750.33	$0 * 10^{47}$
200	1325.12	$0 * 10^{47}$
220	540.34	$-3 * 10^{59}$
240	2250.49	$0 * 10^{40}$
260	2715.47	$0 * 10^{20}$
280	5625.93	$-1.249915... * 10^{12}$

Own computations: (subdivision, $Q_{2\times m}$)

after 79.4 s:

$$\int_0^{2.5} ... \in (-1.249915...10^{12}, -1.249915...10^{12})$$

after 94.6 s:

$$\int_0^{\infty} ... \in (3771213.96, 3771213.99)$$

8. Further Improvement

Theorem 1. *Computations with d correct digits, our integrand (analytic, evaluation cost $\sim d$ multiplications), integration with "certain" formulas*

$$Q_{n\times m}[f] = \sum_{\nu=1}^{n} \sum_{\mu=0}^{m-1} a_{\nu,\mu} \frac{f^{(\mu)}(x_\nu)}{\mu!}.$$

Then, $m, n \sim \sqrt{d}$ *is optimal.*

Theorem 2. x_ν: zeros of T_n, $n = 2^k$. **Then,** the $a_{\nu,\mu}$-s can be calculated with $O(n \cdot m^3 + mn \ln n)$ arithmetic operations.

Application as in Theorem 1 costs $\sim nm^2$