

[Help](#)

```
/* Control Variables Kemna & Vorst Monte Carlo simulation for a Call or Put
   Fixed Asian option. In the case of Monte Carlo simulation, the program
   provides estimations for price and delta with a confidence interval. In
   the case of Quasi-Monte Carlo simulation, the program just provides
   estimations for price and delta. */

#include "
href../../mod/bs1d/bs1d_pad/bs1d_pad_h_src.pdfbs1d_pad.h"
#include "
href../../common/enums_h_src.pdfenums.h"

/* ----- */
/* Calculus of the average  $A'(T_0, T)$  and  $C'(T_0, T)$  of the asian option with one
   of the 3 different schemes One iteration of the Monte Carlo method called
   from the "FixedAsian_KemanVorst" function */
/* ----- */
static void Simul_StockAndAverage_KemnaVorst(int scheme, int generator, int step
{
    double integral, w_t, w_t_1, S_t, current_t, g1, g2;
    double h = T / step_number;
    double sqrt_h = sqrt(h);
    double trend = (r - divid) - 0.5 * SQR(sigma);
    double int2, bb, intt, intt2;
    int i;

    /*Initialisation*/
    *theta = MAX(0., (-trend + 2.*(K / x - 1. / T)) / sigma);
    integral = 0.0;
    intt = 0.0;
    intt2 = 0.0;
    w_t = 0.0;
    current_t = 0.0;
    int2 = 0.0;

    /* Average Computation */
    /* Application of one of the three schemes */
    /* Scheme 1 : Rieman sums */
    if ((scheme != 2) && (scheme != 3))
    {
```

```

/* Simulation of M gaussian variables according to the generator type,
   that is Monte Carlo or Quasi Monte Carlo. */
g1 = pnl_rand_gauss(step_number, CREATE, 0, generator);

for (i = 0 ; i < step_number ; i++)
{
    S_t = exp(trend * current_t + sigma * (w_t + (*theta) * current_t));
    integral += x * S_t;
    intt += current_t * x * S_t;
    intt2 += current_t * current_t * x * S_t;
    int2 += w_t;

    current_t += h;
    /* gaussian value from the table Gaussians */
    g1 = pnl_rand_gauss(step_number, RETRIEVE, i, generator);
    w_t += sqrt_h * g1;
}
}
else
{
    /* Scheme 2 : Trapezoidal method */
    if (scheme == 2)
    {
        /* Simulation of M gaussian variables according to the generator type,
           that is Monte Carlo or Quasi Monte Carlo. */
        g1 = pnl_rand_gauss(step_number, CREATE, 0, generator);

        for (i = 0; i < step_number; i++)
        {
            /* gaussian value from the table Gaussians */
            g1 = pnl_rand_gauss(step_number, RETRIEVE, i, generator);

            w_t_1 = sqrt_h * g1 + w_t;
            S_t = exp(trend * current_t + sigma * (w_t + (*theta) * current_t)
            integral += x * S_t * (1 + (r - divid) * h / 2. + sigma * (w_t_1 -
            intt += current_t * x * S_t * (1 + (r - divid) * h / 2. + sigma * (
            intt2 += current_t * current_t * x * S_t * (1 + (r - divid) * h / 2.
            int2 += (w_t + w_t_1) / 2.;
            current_t += h;
            w_t = w_t_1;
        }
    }
}

```

```

    }
else
{
    /* Scheme 3 : Brownian Bridge method */

    /* Simulation of 2M gaussian variables according to the generator type
       that is Monte Carlo or Quasi Monte Carlo. */
    g1 = pnl_rand_gauss(2 * step_number, CREATE, 0, generator);

    for (i = 0; i < step_number; i++)
    {
        g1 = pnl_rand_gauss(step_number, RETRIEVE, 2 * i, generator);
        w_t_1 = sqrt_h * g1 + w_t;

        g2 = pnl_rand_gauss(step_number, RETRIEVE, (2 * i) + 1, generator);
        bb = (w_t + w_t_1) / 2. + g2 * sqrt(h / 6.);
        S_t = exp(trend * current_t + sigma * (w_t + (*theta) * current_t)

        integral += x * S_t * (1 + (r - divid) * h / 2. + sigma * (bb - w_
        intt += current_t * x * S_t * (1 + (r - divid) * h / 2. + sigma * (
        intt2 += current_t * current_t * x * S_t * (1 + (r - divid) * h / 2.

        int2 += bb;
        current_t += h;
        w_t = w_t_1;
    }
}

/* Final average A'(T0,T) */
*average = integral / step_number;
*averaget = intt / step_number;
*averaget2 = intt2 / step_number;

/* Final average C'(T0,T) */
*controle = x * exp(trend * T / 2. + sigma * (int2 + (*theta) * SQR(T) / (2.*h

/* Final brownian */
*b_t = w_t;

return;

```

```
}
```

```
/* -----*/
/* Pricing of a asian option by the Monte Carlo Kemna & Vorst method
   Estimator of the price and the delta.
   s et K are pseudo-spot and pseudo-strike. */
/* ----- */
static int FixedAsian_KemnaVorst(double s, double K, double time_spent, NumFunc
{
    long i;
    double d1, d2, N_d1, N_d2, S_T;
    double price_Q, delta_Q;
    double average, averaget, averaget2, controle;
    double price_sample, price_sample1 , price_sample2 , delta_sample = 0., mean_p
    int init_mc;
    int simulation_dim;
    double alpha, z_alpha, theta, w_t;

    /* Value to construct the confidence interval */
    alpha = (1. - confidence) / 2.;
    z_alpha = pnlnv_cdfnor(1. - alpha);

    /*Initialisation*/
    mean_price = 0.0;
    mean_delta = 0.0;
    var_price = 0.0;
    var_delta = 0.0;
    /* Size of the random vector we need in the simulation */
    if (scheme == 3)
        simulation_dim = 2 * M;
    else
        simulation_dim = M;

    /* Computation of the price and the delta for the term Q with the control vari
    d1 = (log(s / K) + (r - divid + sigma * sigma / 6.0) * (t / 2.)) / (sigma * sq
    d2 = d1 - sigma * sqrt(t / 3.);
    /* Put case */
    if ((p->Compute) == &Put_OverSpot2)
    {
```

```

        N_d1 = cdf_nor(-d1);
        N_d2 = cdf_nor(-d2);
        price_Q = exp(-r * t) * (K * N_d2 - s * exp((r - divid - sigma * sigma / 6.0 * (t / 2.)) * N_d1 * (1 -
    }
/* Call case */
/*if ( (p->Compute) == &Call_OverSpot2)*/
else
{
    N_d1 = cdf_nor(d1);
    N_d2 = cdf_nor(d2);
    price_Q = exp(-r * t) * (s * exp((r - divid - sigma * sigma / 6.0 * (t / 2.)) * N_d1 * (1 -
    delta_Q = exp((r - divid - sigma * sigma / 6.0 * (t / 2.)) * N_d1 * (1 -
}

/* MC sampling */
init_mc = pnl_rand_init(generator, simulation_dim, nb);
/* Test after initialization for the generator */
if (init_mc == OK)
{

    /* Begin of the N iterations */
    for (i = 1; i <= nb; i++)
    {
        /* Price */
        (void)Simul_StockAndAverage_KemnaVorst(scheme, generator, M, t, s, r,
        price_sample1 = (p->Compute)(p->Par, s, average);
        price_sample2 = (p->Compute)(p->Par, s, controle);
        price_sample = (price_sample1 - price_sample2) * exp(-theta * (theta *
        /*price_inc1_p=(p->Compute)(p->Par, s*(1.+inc), (1.+inc)*average);
        price_inc1_m=(p->Compute)(p->Par, s*(1.-inc), (1.-inc)*average);
        price_inc2_p= (p->Compute)(p->Par, s*(1.+inc), (1.+inc)*controle);
        price_inc2_m= (p->Compute)(p->Par, s*(1.-inc), (1.-inc)*controle);
        price_inc_p= price_inc1_p - price_inc2_p;
        price_inc_m= price_inc1_m - price_inc2_m;*/

        /* Delta */
        if (delta_met == 1)
        {
            delta_sample = 0.0;

```

```

/* According to the Call formula */
if (price_sample1 > 0.0)
    /*delta_sample+=(price_inc1_p-price_inc1_m)/(2.0*s*inc);*/
    delta_sample += exp(-theta * (theta * t / 2. + w_t)) * (1 - time
if (price_sample2 > 0.0)
    /*delta_sample-=(1.0-time_spent)*(price_inc2_p-price_inc2_m)/(2.
    delta_sample -= exp(-theta * (theta * t / 2. + w_t)) * (1 - time
}
if (delta_met == 2)
{
    S_T = s * exp(sigma * w_t + t * (r - divid - SQR(sigma) / 2.));
    delta_sample = 0.0;
    /* According to the Call formula */
    delta_sample += price_sample1 * exp(-theta * (theta * t / 2. + w_t
    /*delta_sample-=(1.0-time_spent)*(price_inc2_p-price_inc2_m)/(2.0*
    delta_sample *= (1 - time_spent);
}
if (delta_met == 3)
{
    S_T = s * exp(sigma * w_t + t * (r - divid - SQR(sigma) / 2.));
    delta_sample = 0.0;
    /* According to the Call formula */
    delta_sample += price_sample1 * exp(-theta * (theta * t / 2. + w_t
    /*delta_sample-=(1.0-time_spent)*(price_inc2_p-price_inc2_m)/(2.0*
    delta_sample *= (1 - time_spent);
}
/* Sum */
mean_price += price_sample;
mean_delta += delta_sample;

/* Sum of squares */
var_price += SQR(price_sample);
var_delta += SQR(delta_sample);
}
/* End of the N iterations */

/* Price estimator */
*ptprice = (mean_price / (double)nb);
*pterror_price = exp(-r * t) * sqrt(var_price / (double)nb - SQR(*ptprice)
*ptprice = exp(-r * t) * (*ptprice) + price_Q;

```

```

/* Price Confidence Interval */
*inf_price = *ptprice - z_alpha * (*pterror_price);
*sup_price = *ptprice + z_alpha * (*pterror_price);

/* Delta estimator */
*ptdelta = exp(-r * t) * (mean_delta / (double)nb);
/* Put Case */
if ((p->Compute) == &Put_OverSpot2)
    *ptdelta *= (-1);
*pterror_delta = sqrt(exp(-2.0 * r * t) * (var_delta / (double)nb - SQR(*p
if (delta_met == 1)
    *ptdelta += exp(-r * t) * (delta_Q);

/* Delta Confidence Interval */
*inf_delta = *ptdelta - z_alpha * (*pterror_delta);
*sup_delta = *ptdelta + z_alpha * (*pterror_delta);
}
return init_mc;
}

```

```

int CALC(MC_FixedAsian_KemnaVorst)(void *Opt, void *Mod, PricingMethod *Met)
{
    TYPEOPT *ptOpt = (TYPEOPT *)Opt;
    TYPEMOD *ptMod = (TYPEMOD *)Mod;

    double T, t_0, T_0;
    double r, divid, time_spent, pseudo_strike, true_strike, pseudo_spot;
    int return_value;

    r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
    divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);

    T = ptOpt->Maturity.Val.V_DATE;
    T_0 = ptMod->T.Val.V_DATE;
    t_0 = (ptOpt->PathDep.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE;
    time_spent = (T_0 - t_0) / (T - t_0);

    if (T_0 < t_0)

```

```

{
    Fprintf(TOSCREEN, "T_0 < t_0, untreated case\ n\ n\ n");
    return_value = WRONG;
}

/* Case t_0 <= T_0 */
else
{
    pseudo_spot = (1. - time_spent) * ptMod->S0.Val.V_PDOUBLE;
    pseudo_strike = (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE - ti

    true_strike = (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE;

    (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE = pseudo_strike;

    if (pseudo_strike <= 0.)
    {
        Fprintf(TOSCREEN, "FORMULE ANALYTIQUE\ n\ n\ n");
        return_value = Analytic_KemnaVorst(pseudo_spot,
                                           pseudo_strike,
                                           time_spent,
                                           ptOpt->PayOff.Val.V_NUMFUNC_2,
                                           T - T_0,
                                           r,
                                           divid,
                                           &(Met->Res[0].Val.V_DOUBLE),
                                           &(Met->Res[1].Val.V_DOUBLE));
    }
    else
        return_value = FixedAsian_KemnaVorst(pseudo_spot,
                                              pseudo_strike,
                                              time_spent,
                                              ptOpt->PayOff.Val.V_NUMFUNC_2,
                                              T - T_0,
                                              r,
                                              divid,
                                              ptMod->Sigma.Val.V_PDOUBLE,
                                              Met->Par[2].Val.V_LONG,
                                              Met->Par[0].Val.V_INT2,

```



```

Met->Par[3].Val.V_ENUM.value,
Met->Par[1].Val.V_ENUM.value,
Met->Par[4].Val.V_DOUBLE,
Met->Par[5].Val.V_ENUM.value,
&(Met->Res[0].Val.V_DOUBLE),
&(Met->Res[1].Val.V_DOUBLE),
&(Met->Res[2].Val.V_DOUBLE),
&(Met->Res[3].Val.V_DOUBLE),
&(Met->Res[4].Val.V_DOUBLE),
&(Met->Res[5].Val.V_DOUBLE),
&(Met->Res[6].Val.V_DOUBLE),
&(Met->Res[7].Val.V_DOUBLE));

    (ptOpt->PayOff.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE = true_strike;
}
return return_value;
}

static int CHK_OPT(MC_FixedAsian_KemnaVorst)(void *Opt, void *Mod)
{
    if ((strcmp(((Option *)Opt)->Name, "AsianCallFixedEuro") == 0) || (strcmp(((Op
        return OK;

    return WRONG;
}

static PremiaEnumMember delta_method_kv_members[] =
{
    { "Finite Difference", 1},
    { "Malliavin FLLLT", 2},
    { "Malliavin Benhamou", 3},
    { NULL, NULLINT }
};

static DEFINE_ENUM(delta_method_kv, delta_method_kv_members);

static int MET(Init)(PricingMethod *Met, Option *Opt)
{

```

```

int type_generator;
if (Met->init == 0)

{
    Met->init = 1;

    Met->Par[0].Val.V_INT2 = 50;
    Met->Par[1].Val.V_ENUM.value = 0;
    Met->Par[1].Val.V_ENUM.members = &PremiaEnumRNGs;
    Met->Par[2].Val.V_LONG = 20000;
    Met->Par[3].Val.V_ENUM.value = 3;
    Met->Par[3].Val.V_ENUM.members = &PremiaEnumIntegralScheme;
    Met->Par[4].Val.V_DOUBLE = 0.95;
    Met->Par[5].Val.V_ENUM.value = 2;
    Met->Par[5].Val.V_ENUM.members = &delta_method_kv;

}
type_generator = Met->Par[1].Val.V_ENUM.value;

if (pnl_rand_or_quasi(type_generator) == PNL_QMC)
{
    Met->Res[2].Viter = IRRELEVANT;
    Met->Res[3].Viter = IRRELEVANT;
    Met->Res[4].Viter = IRRELEVANT;
    Met->Res[5].Viter = IRRELEVANT;
    Met->Res[6].Viter = IRRELEVANT;
    Met->Res[7].Viter = IRRELEVANT;

}
else
{
    Met->Res[2].Viter = ALLOW;
    Met->Res[3].Viter = ALLOW;
    Met->Res[4].Viter = ALLOW;
    Met->Res[5].Viter = ALLOW;
    Met->Res[6].Viter = ALLOW;
    Met->Res[7].Viter = ALLOW;
}

return OK;

```

```
}
```

```
PricingMethod MET(MC_FixedAsian_KemnaVorst) =  
{  
    "MC_FixedAsian_KemnaVorst",  
    { {"TimeStepNumber", INT2, {100}, ALLOW},  
      {"RandomGenerator", ENUM, {100}, ALLOW},  
      {"N iterations", LONG, {100}, ALLOW},  
      {"Integral Scheme", ENUM, {100}, ALLOW},  
      {"Confidence Value", DOUBLE, {100}, ALLOW},  
      {"Delta Method", ENUM, {100}, ALLOW},  
      {" ", PREMIA_NULLTYPE, {0}, FORBID}  
    },  
    CALC(MC_FixedAsian_KemnaVorst),  
    { {"Price", DOUBLE, {100}, FORBID},  
      {"Delta", DOUBLE, {100}, FORBID} ,  
      {"Error Price", DOUBLE, {100}, FORBID},  
      {"Error Delta", DOUBLE, {100}, FORBID} ,  
      {"Inf Price", DOUBLE, {100}, FORBID},  
      {"Sup Price", DOUBLE, {100}, FORBID} ,  
      {"Inf Delta", DOUBLE, {100}, FORBID},  
      {"Sup Delta", DOUBLE, {100}, FORBID} ,  
      {" ", PREMIA_NULLTYPE, {0}, FORBID}  
    },  
    CHK_OPT(MC_FixedAsian_KemnaVorst),  
    CHK_ok,  
    MET(Init)  
};
```