

[Help](#)

```
#include "
href../../../../mod/bs1d/bs1d_doublim/bs1d_doublim_h_src.pdfbs1d_doublim.h"
#include "pnl/pnl_cdf.h"

int MOD_OPT(ChkMix)(Option *Opt, Model *Mod)
{
    TYPEOPT *ptOpt = (TYPEOPT *) (Opt->TypeOpt);
    TYPEMOD *ptMod = (TYPEMOD *) (Mod->TypeModel);
    int status = OK;

    /*Custom*/
    if (ptOpt->Maturity.Val.V_DATE <= ptMod->T.Val.V_DATE)
    {
        Fprintf(TOSCREENANDFILE, "Current date greater than maturity!\ n");
        status += 1;
    };
    if (((ptOpt->LowerLimit.Val.V_NUMFUNC_1)->Compute)((ptOpt->LowerLimit.Val.V_NUMFUNC_1)->Compute))
    {
        Fprintf(TOSCREENANDFILE, "Limit Down greater than spot!\ n");
        status += 1;
    };

    if (((ptOpt->UpperLimit.Val.V_NUMFUNC_1)->Compute)((ptOpt->UpperLimit.Val.V_NUMFUNC_1)->Compute))
    {
        Fprintf(TOSCREENANDFILE, "Limit Up lower than spot!\ n");
        status += 1;
    };
    /*EndCustom*/

    return status;
}

extern PricingMethod MET(MC_OutBaldi);
extern PricingMethod MET(MC_InBaldi);
extern PricingMethod MET(MC_ParisianOut);
extern PricingMethod MET(MC_ParisianIn);
extern PricingMethod MET(AP_Out_Laplace);
extern PricingMethod MET(CF_CallOut_KunitomoIkeda);
extern PricingMethod MET(CF_PutIn_KunitomoIkeda);
```

```

extern PricingMethod MET(CF_PutOut_KunitomoIkeda);
extern PricingMethod MET(FD_Psor_Out);
extern PricingMethod MET(FD_Psor_In);
extern PricingMethod MET(FD_Cryer_Out);
extern PricingMethod MET(FD_Cryer_In);
extern PricingMethod MET(FD_Gauss_In);
extern PricingMethod MET(FD_Gauss_Out);
extern PricingMethod MET(FD_Fem_Out);
extern PricingMethod MET(CF_CallIn_KunitomoIkeda);
extern PricingMethod MET(TR_Ritchken_In);
extern PricingMethod MET(TR_Ritchken_Out);
extern PricingMethod MET(AP_LaplaceDoubleParisian);
extern PricingMethod MET(TR_AGZ);

```

```

PricingMethod *MOD_OPT(methods)[] =
{
    &MET(CF_CallOut_KunitomoIkeda),
    &MET(CF_PutOut_KunitomoIkeda),
    &MET(CF_CallIn_KunitomoIkeda),
    &MET(CF_PutIn_KunitomoIkeda),
    &MET(AP_Out_Laplace),
    &MET(FD_Psor_Out),
    &MET(FD_Psor_In),
    &MET(FD_Cryer_Out),
    &MET(FD_Cryer_In),
    &MET(FD_Gauss_In),
    &MET(FD_Gauss_Out),
    &MET(FD_Fem_Out),
    &MET(TR_Ritchken_In),
    &MET(TR_Ritchken_Out),
    &MET(MC_OutBaldi),
    &MET(MC_InBaldi),
    &MET(MC_ParisianOut),
    &MET(MC_ParisianIn),
    &MET(AP_LaplaceDoubleParisian),
    &MET(TR_AGZ),
    NULL
};

```

```

extern DynamicTest MOD_OPT(test);
DynamicTest *MOD_OPT(tests)[] =

```

```

{
    &MOD_OPT(test),
    NULL
};

Pricing MOD_OPT(pricing) =
{
    ID_MOD_OPT,
    MOD_OPT(methods),
    MOD_OPT(tests),
    MOD_OPT(ChkMix)
};

/* Utility function shared
 *
 *
 */

int MOD_OPT(PutOut_KunitomoIkeda_91_lib)(double s, NumFunc_1 *L, NumFunc_1 *U,
    double t, double r, double divid, double sigma, double *ptprice, double *ptd
{
    double d1, d2, d3, d4, mu1, mu2, mu3, u, l, sum1, sum2, time, k, delta1, delta2;
    int n;

    sum1 = 0.0;
    sum2 = 0.0;
    time = 0.;
    u = (U->Compute)(U->Par, time);
    l = (L->Compute)(L->Par, time);
    k = PayOff->Par[0].Val.V_PDDOUBLE;
    delta1 = 0.0;
    delta2 = 0.0;
    E = 1 * exp(delta2 * t);

    for (n = -5; n <= 5; n++)
    {
        mu1 = 2.*(r - divid - delta2 - (double)n * (delta1 - delta2)) / SQR(sigma);
        mu2 = 2.*(double)n * (delta1 - delta2) / SQR(sigma);
        mu3 = mu1;
    }
}

```

```

    d1 = (log(s * pow(u, 2.0 * (double)n) / (E * pow(l, 2.0 * (double)n))) +
          (r - divid + SQR(sigma) / 2.0) * t) / (sigma * sqrt(t));
    d2 = (log(s * pow(u, 2.0 * n) / (k * pow(l, 2.0 * (double)n))) +
          (r - divid + SQR(sigma) / 2.0) * t) / (sigma * sqrt(t));
    d3 = (log(pow(l, 2.0 * (double)n + 2.) / (E * s * pow(u, 2.0 * (double)n)) +
          (r - divid + SQR(sigma) / 2.0) * t) / (sigma * sqrt(t));
    d4 = (log(pow(l, 2.0 * (double)n + 2.) / (k * s * pow(u, 2.0 * (double)n)) +
          (r - divid + SQR(sigma) / 2.0) * t) / (sigma * sqrt(t));
    sum2 += pow(pow(u / l, (double)n), mu1 - 2.) * pow(l / s, mu2) * (cdf_nor(
          cdf_nor(d2 - sigma * sqrt(t))) -
          pow(pow(l, (double)(n + 1)) / (s * pow(u, (double)n)), mu3 - 2.)
          * (cdf_nor(d3 - sigma * sqrt(t)) - cdf_nor(d4 - sigma * sqrt(t))));
    sum1 += pow(pow(u / l, (double)n), mu1) * pow(l / s, mu2) * (cdf_nor(d1) -
          pow(pow(l, (double)(n + 1)) / (s * pow(u, (double)n)), mu3) * (cdf

}

/*Price*/
*ptprice = k * exp(-r * t) * sum2 - s * exp(-divid * t) * sum1;

/*Delta*/
*ptdelta = 0.0;

return OK;
}

int MOD_OPT(CallOut_KunitomoIkeda_91_lib)(double s, NumFunc_1 *L, NumFunc_1 *U,
{
    double d1, d2, d3, d4, mu1, mu2, mu3, F, u, l, sum1, sum2, time, k, delta1, de
    int n;

    sum1 = 0.0;
    sum2 = 0.0;
    time = 0.;
    u = (U->Compute)(U->Par, time);
    l = (L->Compute)(L->Par, time);
    k = PayOff->Par[0].Val.V_PDOUBLE;
    delta1 = 0.0;
    delta2 = 0.0;
    F = u * exp(delta1 * t);

```

```

for (n = -5; n <= 5; n++)
{
    mu1 = 2.*(r - divid - delta2 - (double)n * (delta1 - delta2)) / SQR(sigma)
    mu2 = 2.*(double)n * (delta1 - delta2) / SQR(sigma);
    mu3 = mu1;
    d1 = (log(s * pow(u, 2.0 * (double)n) / (k * pow(l, 2.0 * (double)n))) +
          (r - divid + SQR(sigma) / 2.0) * t) / (sigma * sqrt(t));
    d2 = (log(s * pow(u, 2.0 * n) / (F * pow(l, 2.0 * (double)n))) +
          (r - divid + SQR(sigma) / 2.0) * t) / (sigma * sqrt(t));
    d3 = (log(pow(l, 2.0 * (double)n + 2.) / (k * s * pow(u, 2.0 * (double)n))
          (r - divid + SQR(sigma) / 2.0) * t) / (sigma * sqrt(t));
    d4 = (log(pow(l, 2.0 * (double)n + 2.) / (F * s * pow(u, 2.0 * (double)n))
          (r - divid + SQR(sigma) / 2.0) * t) / (sigma * sqrt(t));
    sum2 += pow(pow(u / l, (double)n), mu1 - 2.) * pow(l / s, mu2) * (cdf_nor(
          cdf_nor(d2 - sigma * sqrt(t))) -
          pow(pow(l, (double)(n + 1)) / (s * pow(u, (double)n)), mu3 - 2.) *
    sum1 += pow(pow(u / l, (double)n), mu1) * pow(l / s, mu2) * (cdf_nor(d1) -
          pow(pow(l, (double)(n + 1)) / (s * pow(u, (double)n)), mu3) * (cdf
    }

/*Price*/
*ptprice = s * exp(-divid * t) * sum1 - k * exp(-r * t) * sum2;

/*Delta*/
*ptdelta = 0.;

return OK;
}

double MOD_OPT(Boundary_lib)(double s, NumFunc_1 *p, double t, double r, double
{
    double price = 0., delta;

    if ((p->Compute) == &Call)
        pnl_cf_call_bs(s, p->Par[0].Val.V_PDOUBLE, t, r, divid, sigma, &price, &delt
    else if ((p->Compute) == &Put)
        pnl_cf_put_bs(s, p->Par[0].Val.V_PDOUBLE, t, r, divid, sigma, &price, &delta

    return price;
}

```