

[Help](#)

```
#include "bs1d_pad.h"

static double y(double y_min,double i,double h)
{
    return ((y_min)+(i)*(h));
}

static double coth(double x)
{
    return (1/tanh(x));
}

static int Fixed_RodgerShi(double pseudo_stock,
    double pseudo_strike,NumFunc_2 *p,double t,double r,
    double divid,double sigma,int N,int M,double *ptprice,
    double *ptdelta)
{
    int i,j,j0,PriceIndex;
    double y0,y_min,y_max,k,h,boundary;
    double fact1,fact2,fact3,gammaj,alphaj;
    double *a,*b,*P,*S,*alpha,*beta,*gamma;
    double *A,*B,*C;

    /*Memory Allocation*/
    A=(double *)malloc((N+2)*sizeof(double));
    if (A==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    B=(double *)malloc((N+2)*sizeof(double));
    if (B==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    C=(double *)malloc((N+2)*sizeof(double));
    if (C==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    a=(double *)malloc((N+1)*sizeof(double));
    if (a==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    b=(double *)malloc((N+1)*sizeof(double));
    if (b==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    alpha=(double *)malloc((N+1)*sizeof(double));
```

```

if (alpha==NULL)
    return MEMORY_ALLOCATION_FAILURE;
beta=(double *)malloc((N+1)*sizeof(double));
if (beta==NULL)
    return MEMORY_ALLOCATION_FAILURE;
gamma=(double *)malloc((N+1)*sizeof(double));
if (gamma==NULL)
    return MEMORY_ALLOCATION_FAILURE;
P=(double *)malloc((N+1)*sizeof(double));
if (P==NULL)
    return MEMORY_ALLOCATION_FAILURE;
S=(double *)malloc((N+1)*sizeof(double));
if (S==NULL)
    return MEMORY_ALLOCATION_FAILURE;

/*Space Localisation*/
y0=pseudo_strike/pseudo_stock;
y_min=0.;
y_max=6;

/*Time Step*/
k=t/(double)M;

/*Space Step*/
h=(y_max-y_min)/(double)N;

/*Lhs Factor*/
fact1=sigma*sigma/(double)(2*h*h);
fact2=-((r-divid)+sigma*sigma)/(double)(2*h);
fact3=-1/(t*h);
for(j=1;j<N;j++){
    /* Operator L1 Diffusion */
    alpha[j]=fact1*(y(y_min,j-0.5,h)*y(y_min,j-0.5,h));
    beta[j]=-fact1*(y(y_min,j-0.5,h)*y(y_min,j-0.5,h)+y(y_min,j+0.5,h)*y(y_min,j+0.5,h));
    gamma[j]=fact1*(y(y_min,j+0.5,h)*y(y_min,j+0.5,h));

    /* Operator L1 Drift */
    alpha[j]+=-fact2*(y(y_min,j-0.5,h));

```

```

    beta[j]+=fact2*(-y(y_min,j+0.5,h)+y(y_min,
j-0.5,h));
    gamma[j]+=fact2*(y(y_min,j+0.5,h));

    /* Operator L2 */
    gammaj=2*h/(double)(t*sigma*sigma*y(y_min,
j,h)*y(y_min,j,h));
    alphaj=0.5*(1+coth(gammaj/(double)2))-1/(
double)gammaj;

    /* 1/2 < alphaj < 1 : 1/2 = centered ; 1 =
decentered */

    alphaj=0.5;

    alpha[j]+=-fact3*alphaj;
    beta[j]+=fact3*(2*alphaj-1);
    gamma[j]+=fact3*(1-alphaj);

alpha[j]/=2.;
beta[j]/=2.;
gamma[j]/=2.;

    /* + Time term */
    beta[j]+=-1.0/k;
}

/* To prepare the inversion */
for(PriceIndex=1;PriceIndex<N;PriceIndex++){
    A[PriceIndex]=alpha[PriceIndex];
    B[PriceIndex]=beta[PriceIndex];
    C[PriceIndex]=gamma[PriceIndex];
}

for(PriceIndex=N-2;PriceIndex>=1;PriceIndex--)
)
    B[PriceIndex]=B[PriceIndex]-C[PriceIndex]*
A[PriceIndex+1]/B[PriceIndex+1];
for(PriceIndex=1;PriceIndex<N;PriceIndex++)
    A[PriceIndex]=A[PriceIndex]/B[PriceIndex];
for(PriceIndex=1;PriceIndex<N-1;PriceIndex++)

```

```

    C[PriceIndex]=C[PriceIndex]/B[PriceIndex+1]
;

/*Final Condition*/
if ((p->Compute) == &Call_OverSpot2)
    for(j=0;j<=N;j++) /* CALL */
P[j]=0.0;
else
    for(j=0;j<=N;j++) /* PUT */
P[j]=y(y_min,j,h);

/*Finite Difference Cycle Implicite Schema :
back in time*/
for(i=M-1;i>=0;i--)
{
/*Boundary Value*/

if ((p->Compute) == &Call_OverSpot2) {
    if ((r-divid)==0.){
        boundary=(t-k*(double)i)/t;
    }
    else
        boundary=(1.0/((r-divid)*t))*(1.0-exp(-(
r-divid)*(t-k*(double)i))));}
else{
    boundary=exp(-(r-divid)*(t-k*(double)i))*y
_max;
    if ((r-divid)==0.){
        boundary--=(t-k*(double)i)/t;
    }
    else
        boundary--=(1.0/((r-divid)*t))*(1.0-exp(-(
r-divid)*(t-k*(double)i))));}

/*set Rhs */
for(j=1;j<N;j++)
    S[j]=-P[j]/k;

/* Dirichlet */
if ((p->Compute) == &Call_OverSpot2) /*

```

```

CALL */
    S[1]-=alpha[1]*boundary;
else /* PUT */
    S[N-1]-=gamma[N-1]*boundary;

    for(j=1;j<N;j++)
        S[j]-=alpha[j]*P[j-1]+(beta[j]+1.0/k)*P[
j]+gamma[j]*P[j+1];

/*Solve the system*/
for(PriceIndex=N-2;PriceIndex>=1;PriceIndex-
-)
    S[PriceIndex]=S[PriceIndex]-C[PriceI
ndex]*S[PriceIndex+1];

P[1] =S[1]/B[1];

for(PriceIndex=2;PriceIndex<=N-1;PriceIndex+
+)
    P[PriceIndex]=S[PriceIndex]/B[PriceI
ndex]-A[PriceIndex]*P[PriceIndex-1];

if ((p->Compute) == &Call_OverSpot2){ /*
CALL */
    P[0]=boundary; /* et P[N]=0 */}
else{ /* PUT */
    P[N-1]=boundary; /* et P[0]=0 */}
}

j0=(int)floor((y0-y_min)/h);

/*Price*/
*ptprice=exp(-divid*t)*pseudo_stock*(P[j0]+(
P[j0+1]-P[j0])*(y0-y(y_min,j0,h))/h);

/*Delta */
*ptdelta=exp(-divid*t)*(P[j0]-y0*(P[j0+1]-P[
j0-1]))/(2.*h));

/*Memory Desallocation*/
free(a);

```

```

        free(b);
        free(alpha);
        free(beta);
        free(gamma);
        free(A);
        free(B);
        free(C);
        free(P);
        free(S);
    /*    free(Obst);*/

    return OK;
}

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

    int return_value;
    double r,divid,time_spent,pseudo_spot,pseudo_
        strike;
    double t_0, T_0;

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

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

    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
    {

```

```

time_spent=(ptMod->T.Val.V_DATE-(ptOpt->PathDep
.Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE)/
(ptOpt->Maturity.Val.V_DATE-(ptOpt->PathDep.
Val.V_NUMFUNC_2)->Par[0].Val.V_PDOUBLE);
pseudo_spot=(1.-time_spent)*ptMod->S0.Val.V_PD
OUBLE;
pseudo_strike=(ptOpt->PayOff.Val.V_NUMFUNC_2)->
Par[0].Val.V_PDOUBLE-time_spent*(ptOpt->PathDep.Val
.V_NUMFUNC_2)->Par[4].Val.V_PDOUBLE;

if (pseudo_strike<=0.)
{
    Fprintf(TOSCREEN,"ANALYTIC FORMULA{n{n{n"});
    return_value=Analytic_KemnaVorst(pseudo_spot
,pseudo_strike,time_spent,ptOpt->PayOff.Val.V_
NUMFUNC_2, ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.V_DA
TE,r,divid,&(Met->Res[0].Val.V_DOUBLE),&(Met->Res[
1].Val.V_DOUBLE));
}
else
    return_value=Fixed_RodgerShi(pseudo_spot,pseu
do_strike,ptOpt->PayOff.Val.V_NUMFUNC_2,ptOpt->
Maturity.Val.V_DATE-ptMod->T.Val.V_DATE,r,divid,pt
Mod->Sigma.Val.V_PDOUBLE,Met->Par[0].Val.V_INT2,
Met->Par[1].Val.V_INT2,&(Met->Res[0].Val.V_DOUBLE),
&(Met->Res[1].Val.V_DOUBLE));
}
return return_value;
}

int CHK_OPT(FD_FixedAsian_RodgerShi)(void *Opt,
void *Mod)
{
    if ( (strcmp(((Option*)Opt)->Name,"
AsianCallFixedEuro")==0) || (strcmp( ((Option*)Opt)->Name,"
AsianPutFixedEuro")==0) )
        return OK;
    return WRONG;
}

static int MET(Init)(PricingMethod *Met)

```

```

{
static int first=1;

if (first)
{
Met->Par[0].Val.V_INT2=1000;
Met->Par[1].Val.V_INT2=1000;

first=0;
}

return OK;
}

PricingMethod MET(FD_FixedAsian_RodgerShi)=
{
"FD_FixedAsian_RodgerShi",
{{"SpaceStepNumber",INT2,2000,ALLOW },{"
TimeStepNumber",INT2,1000,ALLOW},{ " ",END,0,FORBID}}
,
CALC(FD_FixedAsian_RodgerShi),
{{"Price",DOUBLE,100,FORBID},{ "Delta",
DOUBLE,100,FORBID} ,{" ",END,0,FORBID}},
CHK_OPT(FD_FixedAsian_RodgerShi),
CHK_ok,
MET(Init)
};

```

## References