

[Help](#)

```
#include "bs1d_std.h"

static int KamradRitchken_91(int am,double s,
    NumFunc_1 *p,double t,double r,double divid,double sigma,
    int N,double lambda, double *ptprice,double *ptdelta)
{
    int i,j,npoints;
    double h,pu,pm,pd,z,u,d,stock,upperstock;
    double *P,*iv;

    npoints=2*N+1;
    /*Price, intrinsic value arrays*/
    P=(double *)malloc(npoints*sizeof(double));
    if (P==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    iv=(double *)malloc(npoints*sizeof(double));
    if (iv==NULL)
        return MEMORY_ALLOCATION_FAILURE;

    /*Up and Down factors*/
    h=t/(double) N;
    u=exp(lambda*sigma*sqrt(h));
    d=1./u;

    /*Discounted Probability*/
    z=(r-divid)-SQR(sigma)/2.;
    pu=(1./(2.*SQR(lambda))+z*sqrt(h)/(2.*lambda*sigma));
    pm=(1.-1./SQR(lambda));
    pd=(1.-pu-pm);
    pu*=exp(-r*h);
    pm*=exp(-r*h);
    pd*=exp(-r*h);

    /*Intrinsic value initialisation and terminal values*/
    upperstock=s;
    for (i=0;i<N;i++)
        upperstock*=u;
```

```
stock=upperstock;

for (i=0;i<npoints;i++)
{
    iv[i]=(p->Compute)(p->Par,stock);
    P[i]=iv[i];
    stock*=d;
}

/*Backward Resolution*/
for (i=1;i<=N-1;i++)
{
    npoints-=2;
    for (j=0;j<npoints;j++)
    {
        P[j] = pu*P[j]+pm*P[j+1]+pd*P[j+2];
        if (am)
            P[j] = MAX(iv[j+i],P[j]);
    }
}

/*Delta*/
*ptdelta=(P[0]-P[2])/(s*u-s*d);

/*First time step*/
P[0]= pu*P[0]+ pm*P[1]+pd*P[2];
if (am)
    P[0]=MAX(iv[N],P[0]);

/*Price*/
*ptprice=P[0];

/*Memory desallocation*/
free(P);
free(iv);

return OK;
}
```

```

int CALC(TR_KamradRitchken)(void *Opt,void *Mod,
    PricingMethod *Met)
{
    TYPEOPT* ptOpt=( TYPEOPT*)Opt;
    TYPEMOD* ptMod=( TYPEMOD*)Mod;
    double r,divid;

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

    return KamradRitchken_91(ptOpt->EuOrAm.Val.V_
        BOOL,ptMod->S0.Val.V_PDOUBLE,ptOpt->PayOff.Val.V_
        NUMFUNC_1,
        ptOpt->Maturity.Val.V_DATE-ptMod->T.Val.
        V_DATE,
        r,divid,ptMod->Sigma.Val.V_PDOUBLE,
        Met->Par[0].Val.V_INT,Met->Par[1].Val.V_
        RGDOUBLE,
        &(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1]
        .Val.V_DOUBLE));
}

int CHK_OPT(TR_KamradRitchken)(void *Opt, void *
    Mod)
{
    Option* ptOpt=(Option*)Opt;
    TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);

    return OK;
}

static int MET(Init)(PricingMethod *Met)
{
    static int first=1;

    if (first)
    {
        Met->Par[0].Val.V_INT2=100;
        Met->Par[1].Val.V_RGDOUBLE12=1.22474;

        first=0;
    }
}

```

```
    }

    return OK;
}

PricingMethod MET(TR\_KamradRitchken)=
{
    "TR\_KamradRitchken",
        {"StepNumber",INT2,100,ALLOW}, {"Lambda",
RGDOUBLE12,1,ALLOW}, {"  ",END,0,FORBID}},
        CALC(TR\_KamradRitchken),
        {"Price",DOUBLE,100,FORBID}, {"Delta",
DOUBLE,100,FORBID}  , {"  ",END,0,FORBID}},
        CHK_OPT(TR\_KamradRitchken),
        CHK_tree,
        MET(Init)
};
```

References