

Help

```
#include "bs1d_std.h"

static int Sor(double s, NumFunc_1 *p, double t,
    double r, double divid, double sigma, int N, int M, double
    theta, double omega, double epsilon, double *ptpric
    e, double *ptdelta)
{
    double k, z, l, h, vv, alpha, beta, gamma, alpha1, bet
    a1, gamma1, x, y, error, norm, upwind_alphacoeff;
    int i, j, Index, loops;
    double *P, *Rhs;

    /*Memory Allocation*/
    if (N%2==1) N++;
    P=(double *)malloc((N+1)*sizeof(double));
    if (P==NULL)
        return MEMORY_ALLOCATION_FAILURE;
    Rhs=(double *)malloc((N+1)*sizeof(double));
    if (Rhs==NULL)
        return MEMORY_ALLOCATION_FAILURE;

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

    /*Space Localisation*/
    z=(r-divid)-SQR(sigma)/2.0;
    l=sigma*sqrt(t)*sqrt(log(1.0/PRECISION))+fa
    bs(z)*t;

    /*Space Step*/
    h=2.0*l/(double)N;

    /*Peclet Condition-Coefficient of diffusion
    augmented */
    vv=0.5*SQR(sigma);
    if ((h*fabs(z))<=vv)
        upwind_alphacoeff=0.5;
    else {
        if (z>0.) upwind_alphacoeff=0.0;
```

```

    else if (z<=0.) upwind_alphacoef=1.0;
}
vv=-z*h*(upwind_alphacoef-0.5);

/*Lhs factor of theta-schema*/
alpha=theta*k*(-vv/(h*h)+z/(2.0*h));
beta=1.0+k*theta*(r+2.*vv/(h*h));
gamma=k*theta*(-vv/(h*h)-z/(2.0*h));

/*Rhs factor of theta-schema*/
alpha1=k*(1.0-theta)*(vv/(h*h)-z/(2.0*h));
beta1=1.0-k*(1.0-theta)*(r+2.*vv/(h*h));
gamma1=k*(1.0-theta)*(vv/(h*h)+z/(2.0*h));

/*Terminal Values*/
x=log(s);
for (i=0;i<=N;i++)
    P[i]=(p->Compute)(p->Par,exp(x-1+(double)
i*h));

/*Finite Difference Cycle*/
for (i=1;i<=M;i++)
{
    /*Init Rhs*/
    for(j=1;j<N;j++)
        Rhs[j]=alpha1*P[j-1]+beta1*P[j]+gamma1*P[j+1];

    /*Sor Cycle*/
    loops=0;
    do
    {
        error=0.;
        norm=0.;

        for(j=1;j<N;j++)
        {
            y=(Rhs[j]-alpha*P[j-1]-gamma*P[j+1])/beta;
            y=P[j]+omega*(y-P[j]);

```

```

        error+=(double)(j+1)*fabs(y-P[j])
;
        norm+=fabs(y);
        P[j]=y;
    }

    if (norm<1.0) norm=1.0;
    error=error/norm;

    loops++;
} while ((error>epsilon) && (loops<MAX
LOOPS));
/*End Sor Cycle*/
}
/*End Finite Difference Cycle*/

Index=(int) floor ((double)N/2.0);

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

/*Delta*/
*ptdelta=(P[Index+1]-P[Index-1])/(2.0*s*h);

/*Memory Desallocation*/
free(P);
free(Rhs);

return OK;
}

int CALC(FD_Sor)(void *Opt,void *Mod,Pricing
Method *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/10
0.);

```

```

        return Sor(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_INT,
            Met->Par[2].Val.V_RGDOUBLE,Met->Par[3].Val.V_RGDOUBLE,Met->Par[4].Val.V_RGDOUBLE,
            &(Met->Res[0].Val.V_DOUBLE),&(Met->Res[1].Val.V_DOUBLE));
    }

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

    if ((opt->EuOrAm).Val.V_BOOL==EURO)
        return OK;

    return WRONG;
}

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

    if (first)
    {
        Met->Par[0].Val.V_INT2=100;
        Met->Par[1].Val.V_INT2=100;
        Met->Par[2].Val.V_RGDOUBLE=0.5;
        Met->Par[3].Val.V_RGDOUBLE=1.5;
        Met->Par[4].Val.V_RGDOUBLE=0.000001;

        first=0;
    }

    return OK;
}

```

```

}
```

```

PricingMethod MET(FD_Sor)=
{
  "FD_Sor",
  {"SpaceStepNumber",INT2,100,ALLOW },{"
TimeStepNumber",INT2,100,ALLOW},
  {"Theta",RGDOUBLE051,100,ALLOW}, {"Omeg
a",RGDOUBLE12,100,ALLOW}, {"Epsilon",RGDOUBLE,100
,ALLOW},{" ",END,0,FORBID}},
  CALC(FD_Sor),
  {"Price",DOUBLE,100,FORBID},{"Delta",
DOUBLE,100,FORBID} ,{" ",END,0,FORBID}},
  CHK_OPT(FD_Sor),
  CHK_psor,
  MET(Init)
};
```

References