

[Help](#)

```
#include "bs1d_stda.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2015+2) //The "#els
static int CHK_OPT(FD_GMWB_BS)(void *Opt, void *Mod)
{
    return NONACTIVE;
}
int CALC(FD_GMWB_BS)(void *Opt, void *Mod, PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

static double r, w0, t, sigma;
static double alpha_m;
static int Ns, Ni, Nj, Nf;
static double gr;
static double *Kt;
static double c;

static double withdraw(double x, double dt_nf, double kv)
{
    if (x <= gr * dt_nf)
        return (x);
    else
        return (x - kv * (x - gr * dt_nf) - c);
}

static double asinh1(double x)
{
    double ret;
    if (x > 0) ret = log(x + sqrt(x * x + 1));
    else ret = -log(-x + sqrt(x * x + 1));
    return ret;
}

static double rtsec(double (*func)(double), double x1, double x2, double xacc)
```

```

int j;
double fl, f, dx, swap, x1, rts;

fl = (*func)(x1);
f = (*func)(x2);
if (fabs(fl) < fabs(f))
{
    rts = x1;
    x1 = x2;
    swap = fl;
    fl = f;
    f = swap;
}
else
{
    x1 = x1;
    rts = x2;
}
for (j = 1; j <= 30; j++)
{
    dx = (x1 - rts) * f / (f - fl);
    x1 = rts;
    fl = f;
    rts += dx;
    f = (*func)(rts);
    if (fabs(dx) < xacc || f == 0.0) return rts;
}
printf("Maximum number of iterations exceeded in rtsec\ n");
return 0.0;
}
#undef MAXIT

/*Compute GMWB Price*/
static double compute_price(double alpha_g)
{
    double dt;
    int Index, PriceIndex;
    double vv, hi, hj, alpha, beta, alpha_c, alpha_b, alpha_f, beta_c, beta_b, bet
    double *A, *B, *C, *P, *P_Old, *vect_w, *vect_a;
    int i, i_w, i_a;
    double price, w_value, a_value;

```

```

double v_max;
double val;
double temp, delta;
double cw, w_max;

w_max = 2000.;
c = 1.e-8; //Fixed cost

A = malloc((Ni + 1) * sizeof(double));
B = malloc((Ni + 1) * sizeof(double));
C = malloc((Ni + 1) * sizeof(double));
P = malloc((Ni + 1) * (Nj + 1) * sizeof(double));
P_Old = malloc((Ni + 1) * (Nj + 1) * sizeof(double));
vect_w = malloc((Ni + 1) * sizeof(double));
vect_a = malloc((Nj + 1) * sizeof(double));

dt = t / (double)Ns;

/*Space Localisation*/
vv = SQR(sigma);

hi = w_max / (double)Ni;
for (PriceIndex = 0; PriceIndex <= Ni; PriceIndex++)
{
    vect_w[PriceIndex] = (double)PriceIndex * hi;
}

//SINH mesh
cw = w0 / 5;
temp = asinh1(-w0 / cw);
delta = asinh1((w_max - w0) / cw - temp) / Ni;
for (PriceIndex = 0; PriceIndex <= Ni; PriceIndex++)
{
    vect_w[PriceIndex] = temp + PriceIndex * delta;
}
vect_w[0] = 0;
for (PriceIndex = 1; PriceIndex < Ni; PriceIndex++)
{
    vect_w[PriceIndex] = w0 + cw * sinh(vect_w[PriceIndex]);
}
vect_w[Ni] = 2.*vect_w[Ni - 1] - vect_w[Ni - 2];

```

```

hj = w0 / (double)Nj;
for (Index = 0; Index <= Nj; Index++)
{
    vect_a[Index] = (double)Index * hj;
}

// selection of coefficients Ai, Bi
A[0] = 0.;
C[0] = 0.;
B[0] = 1. + dt * r;
B[Ni] = 1.;
A[Ni] = 0.;
C[Ni] = 0.;

for (PriceIndex = 1; PriceIndex <= Ni - 1; PriceIndex++)
{
    alpha_c = vv * SQR(vect_w[PriceIndex]) / ((vect_w[PriceIndex] - vect_w[PriceIndex + 1]) * (vect_w[PriceIndex] + vect_w[PriceIndex + 1]));
    beta_c = vv * SQR(vect_w[PriceIndex]) / ((vect_w[PriceIndex] - vect_w[PriceIndex + 1]) * (vect_w[PriceIndex] + vect_w[PriceIndex + 1]));
    alpha_f = vv * SQR(vect_w[PriceIndex]) / ((vect_w[PriceIndex] - vect_w[PriceIndex + 1]) * (vect_w[PriceIndex] + vect_w[PriceIndex + 1]));
    beta_f = vv * SQR(vect_w[PriceIndex]) / ((vect_w[PriceIndex] - vect_w[PriceIndex + 1]) * (vect_w[PriceIndex] + vect_w[PriceIndex + 1]));
    alpha_b = vv * SQR(vect_w[PriceIndex]) / ((vect_w[PriceIndex] - vect_w[PriceIndex + 1]) * (vect_w[PriceIndex] + vect_w[PriceIndex + 1]));
    beta_b = vv * SQR(vect_w[PriceIndex]) / ((vect_w[PriceIndex] - vect_w[PriceIndex + 1]) * (vect_w[PriceIndex] + vect_w[PriceIndex + 1]));
    if ((alpha_c >= 0) && (beta_c >= 0))
    {
        alpha = alpha_c;
        beta = beta_c;
    }
    else
    {
        if (beta_f >= 0)
        {
            alpha = alpha_f;
            beta = beta_f;
        }
        else
        {
            alpha = alpha_b;
            beta = beta_b;
        }
    }
}

```

```

    }

    A[PriceIndex] = -dt * alpha;
    B[PriceIndex] = 1. + dt * (alpha + beta + r);
    C[PriceIndex] = -dt * beta / (B[PriceIndex] - A[PriceIndex] * C[PriceIndex]);
}

/*Terminal Values*/
for (PriceIndex = 0; PriceIndex <= Ni; PriceIndex++)
    for (Index = 0; Index <= Nj; Index++)
        P[PriceIndex + (Ni + 1)*Index] = MAX(vect_w[PriceIndex], (1. - Kt[Ns]) * v

/*Dynamic Programming*/
for (i = 0; i <= Ns - 1; i++)
{
    for (PriceIndex = 0; PriceIndex <= Ni; PriceIndex++)
        for (Index = 0; Index <= Nj; Index++)
            P_Old[PriceIndex + (Ni + 1)*Index] = P[PriceIndex + (Ni + 1) * Index];

    if (i % Nf == 0)
    {
        //Whitdrawal amount optimization

        for (PriceIndex = 0; PriceIndex <= Ni - 1; PriceIndex++)
        {
            for (Index = 0; Index <= Nj; Index++)
            {
                v_max = P_Old[PriceIndex + (Ni + 1) * Index];
                gamma = hj / 2.;
                while (gamma < vect_a[Index])
                {
                    w_value = MAX(vect_w[PriceIndex] - gamma, 0.);
                    i_w = 0;
                    while ((vect_w[i_w] < w_value) && (i_w < Ni)) i_w++;
                    a_value = vect_a[Index] - gamma;
                    i_a = 0;
                    while ((vect_a[i_a] < a_value) && (i_a < Nj)) i_a++;
                    if ((i_w == 0) && (i_a == 0)) v_max = MAX(v_max, P_Old[i_w
                    if ((i_w == 0) && (i_a > 0))
                    {

```

```

        val = P_Old[i_w + (Ni + 1) * (i_a - 1)] + (P_Old[i_w +
v_max = MAX(v_max, val + withdraw(gamma, dt * Nf, Kt[N
    }
if ((i_w > 0) && (i_a == 0))
{
    val = P_Old[i_w - 1 + (Ni + 1) * i_a] + (P_Old[i_w +
v_max = MAX(v_max, val + withdraw(gamma, dt * Nf, Kt[N
    }
if ((i_w > 0) && (i_a > 0))
{
    val = (P_Old[i_w - 1 + (Ni + 1) * (i_a - 1)] * (vect_w
v_max = MAX(v_max, val + withdraw(gamma, dt * Nf, Kt[N
    }
//gamma+=hj/2.;
gamma += 4 * hj;
}
//checking gr*dt as value
if (gr * dt * Nf < vect_a[Index])
{
    gamma = gr * dt * Nf;
w_value = MAX(vect_w[PriceIndex] - gamma, 0.);
i_w = 0;
while ((vect_w[i_w] < w_value) && (i_w < Ni)) i_w++;
a_value = vect_a[Index] - gamma;
i_a = 0;
while ((vect_a[i_a] < a_value) && (i_a < Nj)) i_a++;
if ((i_w == 0) && (i_a == 0)) v_max = MAX(v_max, P_Old[i_w
if ((i_w == 0) && (i_a > 0))
{
    val = P_Old[i_w + (Ni + 1) * (i_a - 1)] + (P_Old[i_w +
v_max = MAX(v_max, val + withdraw(gamma, dt * Nf, Kt[N
    }
if ((i_w > 0) && (i_a == 0))
{
    val = P_Old[i_w - 1 + (Ni + 1) * i_a] + (P_Old[i_w +
v_max = MAX(v_max, val + withdraw(gamma, dt * Nf, Kt[N
    }
if ((i_w > 0) && (i_a > 0))
{
    val = (P_Old[i_w - 1 + (Ni + 1) * (i_a - 1)] * (vect_w
v_max = MAX(v_max, val + withdraw(gamma, dt * Nf, Kt[N

```

```

    }
    }
    //checking Aj as value
    gamma = vect_a[Index];
    w_value = MAX(vect_w[PriceIndex] - gamma, 0);
    i_w = 0;
    while ((vect_w[i_w] < w_value) && (i_w < Ni)) i_w++;
    i_a = 0;
    if (i_w == 0) v_max = MAX(v_max, P_Old[i_w + (Ni + 1) * i_a] +
    if (i_w > 0)
    {
        val = P_Old[i_w - 1 + (Ni + 1) * i_a] + (P_Old[i_w + (Ni +
        v_max = MAX(v_max, val + withdraw(gamma, dt * Nf, Kt[Ns -
    }

    P[PriceIndex + (Ni + 1)*Index] = v_max ;
}
} //monitoring

}

//Forward sweep to solve the tridiagonal matrix equation
for (Index = 0; Index <= Nj; Index++)
{
    for (PriceIndex = 0; PriceIndex <= Ni; PriceIndex++)
        P[PriceIndex + (Ni + 1)*Index] += dt * alpha_m * vect_w[PriceIndex];

    P[(Ni + 1)*Index] = P[(Ni + 1) * Index] / B[0];
    for (PriceIndex = 1; PriceIndex <= Ni - 1; PriceIndex++)
    {
        P[PriceIndex + (Ni + 1)*Index] = (P[PriceIndex + (Ni + 1) * Index]
    }
    P[Ni + (Ni + 1)*Index] = vect_w[Ni] * exp(-(alpha_g) * ((double)i + 1.
    for (PriceIndex = Ni - 1; PriceIndex >= 0; PriceIndex--)
    {
        P[PriceIndex + (Ni + 1)*Index] = P[PriceIndex + (Ni + 1) * Index]
    }

}

```

```

    }
    PriceIndex = 0;
    while ((vect_w[PriceIndex] < w0) && (PriceIndex < Ni)) PriceIndex++;
    if ((PriceIndex == 0) || (PriceIndex == Ni)) price = P[PriceIndex + (Ni + 1) *
    else price = P[PriceIndex - 1 + (Ni + 1) * Nj] + (P[PriceIndex + (Ni + 1) * Nj

price = price - w0;

/*Memory Desallocation*/
free(A);
free(B);
free(C);
free(P);
free(vect_w);
free(vect_a);
free(P_Old);

/*Price*/
return price;
}

////////////////////////////////////

static int FD_gmwb_bs(double w0_fixed, double t_fixed, double r_fixed, double si
{
    int i, j;
    double pr1, pr2, pracc, val, dt;

    Ns = (int)(t_fixed * n_step_for_year);
    dt = t_fixed / (double)Ns;
    Ni = Ni_fixed;
    Nj = Nj_fixed;
    t = t_fixed;
    w0 = w0_fixed;
    r = r_fixed;
    sigma = sigma_fixed;
    alpha_m = alpha_m_fixed;
    gr = gr_fixed;

```



```

Nf = (int)(Ns / N_monit);

//Surrender charges
Kt = malloc((Ns + 1) * sizeof(double));

//Surrender fee Kt
for (i = 0; i <= Ns; i++)
{
    j = 0;
    while ((pnl_vect_get(timesKt_fixed, j) < (double)i * dt) && (j <= 5))
        j++;

    Kt[i] = pnl_vect_get(Kt_fixed, j);
}

pracc = 1.e-8;
pr1 = 0.;
pr2 = 0.1;
val = rtsec(compute_price, pr1, pr2, pracc);

/*Price*/
*ptprice = val;

free(Kt);

return OK;
}

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

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

    return FD_gmwb_bs(ptMod->S0.Val.V_PDOUBLE, ptOpt->Maturity.Val.V_DATE - ptMod->
}

static int CHK_OPT(FD_GMWB_BS)(void *Opt, void *Mod)
{

```

```

    if ((strcmp(((Option *)Opt)->Name, "GMWB") == 0))
        return OK;
    else
        return WRONG;
}

#endif //PremiaCurrentVersion

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
    if (Met->init == 0)
    {
        Met->init = 1;
        Met->HelpFilenameHint = "fd_gmwb_bs";
        Met->Par[0].Val.V_INT2 = 10;
        Met->Par[1].Val.V_INT2 = 100;
        Met->Par[2].Val.V_INT2 = 100;
    }

    return OK;
}

PricingMethod MET(FD_GMWB_BS) =
{
    "FD_GMWB_BS",
    { {"Timestep_for_year", INT2, {100}, ALLOW}, {"SpaceStepNumber W", INT2, {100},
        {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CALC(FD_GMWB_BS),
    { {"Fair fee", DOUBLE, {100}, FORBID},
        {" ", PREMIA_NULLTYPE, {0}, FORBID}
    },
    CHK_OPT(FD_GMWB_BS),
    CHK_split,
    MET(Init)
};

```