```
--> /* 7  Black-Scholes 公式 */

--> /* ［数値処理］→［自動的に数値で出力］*/
    if numer#false then numer:false else numer:true;
    /*浮動小数点の桁数=7に指定 */
    fpprec : 7;
    /* 浮動小数点表示桁数を7に指定*/
    fpprintprec:7;

--> /* 7.3 Black-Scholes オプション価格 */

--> black_scholes_1(S,K,r,sigma,T):=
    (
    d1 : (log (S/K) + (r+sigma^2/2) *T) / (sigma*sqrt (T)),
    d2 : d1 - sigma*sqrt (T),
    load(distrib),
    C0 : S*cdf_normal(d1, 0, 1) - exp (-r*T) *K*cdf_normal (d2,0,1),
    print( " コールオプション価格= ", C0)
    );

--> black_scholes_1 (100,100,0.01,0.2,1);

--> /* BS式によるコール価格とプットコールパリティによるプット価格 */

--> black_scholes_2(S,K,r,sigma,T):=
    (
    d1 : (log (S/K) + (r+sigma^2/2) *T) / (sigma*sqrt (T)),
    d2 : d1 - sigma*sqrt (T),
    load(distrib),
    C0 : S*cdf_normal(d1, 0, 1) - exp (-r*T) *K*cdf_normal (d2,0,1),
    P0 : -S+exp(-r*T)*K+C0,
    matrix(
    ["コールオプション価格", "プットオプション価格"],
    [C0, P0]
    )
    );

--> black_scholes_2 (100,100,0.01,0.2,1);

--> / * 7.4 インプライドボラティリティ */
```

```
--> /*Black-Scholes 公式を計算する関数*/
    black_scholes(S,K,r,sigma,T):=
    (
    d1 : (log (S/K) + (r+sigma^2/2) *T) / (sigma*sqrt (T)),
    d2 : d1 - sigma*sqrt (T),
    load(distrib),
    C0 : S*cdf_normal(d1, 0, 1) - exp (-r*T) *K*cdf_normal (d2,0,1)
    );

--> /*与えられた価格とブラックショールズ公式価格との誤差を計算する関数*/
    err(S,K,r,sigma,T,MktPrice):=
    abs (MktPrice-black_scholes (S,K,r,sigma,T));

--> /* 最小化と根のためのパッケージ*/
    load(minpack);

--> /* minpack_lsquares (flist, varlist, guess) */
    /*関数リスト flistの平方和を最小化する変数リスト varlistの値を求める．*/
    /* guessは最適点の初期推測値リスト */
    /* 結果を長さ３のリストとして返し， */
    /*一番目は解; 二番目は平方の和， */
    /*三番目はアルゴリズムの成功を示し, 0は不適切な入力パラメータを表わす
    （それ以外はmanual参照） */

--> minpack_lsquares([err(100, 100, 0.01, sigma, 1, 8.43)], [sigma], [0.25]);

--> /* 限定メモリ準Newton (BFGS)アルゴリズムによる最小化問題解法パッケージ*/
    load (lbfgs);
```

```
--> /*lbfgs (FOM, X, X0, epsilon, iprint)  */
    /*変数リスト Xついて，初期見積もり値リストX0から始めて 、
      norm(grad(FOM)) < epsilon*max(1, norm(X))となる
      関数FOM最小化の解を求める*/
    /*  gradは FOMの多変数 Xに関する勾配 */
    /* normはノルム */
    /* iprintは lbfgsが印字する進捗メッセージを制御するリスト */
    /* iprint[1] < 0 進捗メッセージなし */
    /* iprint[1] = 0 最初と最後の繰り返しでメッセージ */
    /* iprint[1] > 0 毎iprint[1]回の繰り返してメッセージを印字 */
    /* iprint[2] = 0 */
    /* 繰り返し回数、 FOMの評価回数、 FOMの値、 FOMの勾配のノルム、
      ステップ長を印字 */
    /* iprint[2] = 1 */
    /* iprint[2] = 0に加えて、 X0と X0で評価された FOMの勾配を印字 */
    /* iprint[2] = 2 */
    /* iprint[2] = 1に加えて、 繰り返しそれぞれで Xの値を印字  */
    /* iprint[2] = 3 */
    /* iprint[2] = 2に加えて、 繰り返しそれぞれで FOMの勾配を印字 */
    /* lbfgsが印字する列は以下の通り  */
    /* I  */
    /* 繰り返し回数 */
    /* NFN */
    /* 性能指標の評価回数 */
    /* FUNC  */
    /* 最も最近の直線探索の最後での性能指標の値 */
    /* GNORM */
    /* 最も最近の直線探索の最後での性能指標の勾配のノルム */
    /* STEPLENGTH */
    /* 探索アルゴリズムの内部パラメータ */

--> lbfgs(err(100,100,0.01,sigma,1,8.43)^2, [sigma], [0.25], 1e-7, [0,0]);

--> /*K と MktPrice にはベクトルを与える*/
    K_sample :[80,90,100,110,120];
    Mkt_sample :[22.75,15.1,8.43,4.72,3.28];

--> err(100, K_sample, 0.01, sigma, 1, Mkt_sample);

--> minpack_lsquares(err(100, K_sample, 0.01, sigma, 1, Mkt_sample), [sigma], [0.25]);
```

```
--> err1(S,K,r,sigma,T,MktPrice):=
    (
     /*MktPrice と K はベクトル*/
    sum((MktPrice[i]-black_scholes (S,K[i],r,sigma,T))^2, i, 1, length(MktPrice))
    );
```

```
--> /* 限定メモリ準Newton (BFGS)アルゴリズムによる最小化問題解法パッケージ*/
    load (lbfgs);
```

```
--> lbfgs(err1(100,K_sample,0.01,sigma,1,Mkt_sample), [sigma], [0.25], 1e-7, [0,0]);
```