```
--> /* 8 モンテカルロシミュレーション */

--> /* ［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;

--> /* 8.1 モンテカルロシミュレーションの基本 */

--> /* モンテカルロ法によるコールオプション・プライシング*/
    call_monte1(S,K,r,sigma,T,n):=
    (
    load(distrib),
    C0 : sum(max(S*exp ((r-0.5*sigma^2) *T+sigma*sqrt (T) *random_normal (0,1,1))[1]
                    -K,0),i, 1, n)
         *exp (-r*T) /n,
    print ("モンテカルロ法による価格 =",C0)
    );

--> call_monte1 (100,100,0.01,0.2,1,10000);

--> call_monte1_1(S,K,r,sigma,T,n):=
    (
    load(distrib),
    y: S*exp ((r-0.5*sigma^2) *T+sigma*sqrt (T) *random_normal (0,1,n))-K,
    C0 : exp (-r*T) * sum(if y[i]>0 then y[i] else 0, i, 1, n)/n,
    print ("モンテカルロ法による価格 =",C0)
    );

--> call_monte1_1 (100,100,0.01,0.2,1,10000);

--> /* 8.2 分散減少法 */
```

```
--> /* 8.2.1 対称変量法 */
    call_monte2(S,K,r,sigma,T,n):=
    (
    load(distrib),
    epsi : random_normal (0,1,n),
    C0: sum(
        (max (S*exp ((r-0.5*sigma^2) *T+sigma*sqrt (T) *epsi[i]) -K,0)
        +max (S*exp ((r-0.5*sigma^2) *T-sigma*sqrt (T) *epsi[i]) -K,0)), i, 1, n)
        * exp (-r*T) / (2*n),
    print( " モンテカルロ法による価格= ", C0 )
    );

--> call_monte2 (100,100,0.01,0.2,1,10000);

--> /* 8.2.2 モーメントマッチング法 */
    call_monte3(S,K,r,sigma,T,n):=
    (
    load(distrib),
    x : random_normal (0,1,n) /*n 個の正規乱数を発生させる*/,
    y : (x - mean (x))/std (x),
    z : S*exp ((r-0.5*sigma^2) *T+sigma*sqrt (T) *y) -K,
    /*モンテカルロシミュレーションによるオプション価格を求める*/
    C0 : sum (if 0<z[i] then z[i] else 0, i, 1, n) *exp (-r*T) /n,
    print( " モンテカルロ法による価格= ", C0 )
    );

--> call_monte3 (100,100,0.01,0.2,1,10000);

--> /* 8.3 エキゾチックオプション */

--> /*デジタルオプション*/
    digital_monte(S,K,r,sigma,T,n):=
    (
    load(distrib),
    x : random_normal (0,1,n) /*n 個の乱数を発生させる*/,
    y : (x - mean (x))/std (x) /*2 次モーメントまで合致させるように変換する*/,
    /* ペイオフが正のときのみ，加算値 1 として加算する */

    P: sum(
        if (S*exp ((r-0.5*sigma^2) *T+sigma*sqrt (T) *y[i]) >K) then 1 else 0,
      i, 1, n),
    P : exp (-r*T) *P/n,
    print(" デジタルオプション価格 =", P )
    );
```

```
--> digital_monte (100,100,0.01,0.2,1,10000);

--> /*ルックバックオプション*/
    lookback_option(S,K,r,sigma,T,n,M):=
    (
    load(distrib),
    P:0,
    dt : T/M /*M は観測点の数.満期までを M 個のグリッドに分割する.*/
    /* n はモンテカルロのシミュレーション回数 */,
    for i :1 thru n do
    (
    St :S /* 時点 t における原資産価格. 時点 0 における原資産価格S で初期化 */,
    Smax : S /*Smax は満期までの St のうち最大値を格納. S で初期化 */,
    x:random_normal(0,1,M) /* 観測点の数だけ乱数を発生させる */,
    for j:1 thru M do (
    /* M 個の観測点それぞれにおける St をシミュレーションしていく*/
    St : St *exp ((r-0.5*sigma^2) *dt+sigma*sqrt (dt) *x[j]),
    /* t 時点における価格 St がそれ以前の時点における最大値より大きければ, */
    /* 最大値の情報を更新する */
    if St > Smax then Smax : St
    ),
    P:P+max(Smax-K,0)
    ),
    P : exp (-r*T)*P/n,
    print ( " ルックバックオプション価格= ", P )
    );

--> lookback_option (100,100,0.01,0.2,1,10000,100);

--> /* 多資産型オプションの PV 計算について */
```

```
--> /*バスケットオプション*/
    basket_option(S,K,r,Sigma,T,n):=
    (
    load(distrib),
    /* S は対象資産数を含むベクトルとする */
    m : length (S) /* 原資産数 m を定義する */,
    /* Sigma の各行のユークリッドノルムの二乗を計算する */
    v : makelist(1, m).transpose(Sigma^2),
    payoff:
    sum(
    max( (S . exp ((r-0.5*v) *T+(random_normal (0,1,m)*sqrt(T)).transpose(Sigma)) )/m
          -K,0),
    i, 1, n),
    ans: exp(-r*T)*payoff/n
    );

--> Sigma: matrix([0.2,0.4],[0.1,0.5]); S:[100,50]; S:[100, 50];

--> basket_option (S,75,0.01,Sigma,1,10000);

--> /*レインボーオプション*/
    rainbow_option(S,K,r,Sigma,T,n):=
    (
    load(distrib),
    /* S は対象資産数を含むベクトルとする */
    m : length (S) /* 原資産数 m を定義する */,
    /* Sigma の各行のユークリッドノルムの二乗を行ベクトルとして計算する */
    v : makelist(1, m).transpose(Sigma^2),
    /* m次元の標準正規乱数をn個発生させる */
    x : makelist(random_normal (0,1,m),n),
    payoff:
    sum(
     max(
      lmax(
       makelist(
        S[j] * exp (
                    (r-0.5*v[1][j]) *T+Sigma[j].transpose(x[i]*sqrt(T))
                   ) ,j,1,m
                 )
          ) -K, 0
        ), i, 1, n
      ),
    ans: exp(-r*T)*payoff/n
    );
```

```
--> S:[100,100];Sigma: matrix([0.2,0.4],[0.1,0.5]);
    rainbow_option (S,100,0.01,Sigma,1,10000);
```