1. Implement a compound Put option, the option to purchase a European-style Put option, with expiry $T$ and strike price $K$, for price $L$ at time $T_1$ satisfying $0 < T_1 < T$. Use your code to price such an option with parameters $(T, T_1, S_0, K, L, r, v, N) = (2, 1, 90, 95, 4.50, 0.03, 0.15, 20)$. (Hint: modify CRRcc.m.)

Solution: The Octave function CRRcp() differs from CRRcc() at line 25 and in a few updated comments.

```octave
function [W,P] = CRRcp(T, T1, S0, K, L, r, v, N)
% Octave/MATLAB function to price a compound Put in the Cox–Ross–Rubinstein (CRR) binomial model.
% INPUTS: (Example)
% T = expiration time in years (1)
% T1 = choice time; must have 0<T1<T (0.5)
% S0 = spot stock price (90)
% K = stock strike price at expiry T (95)
% L = option strike price at time T1 (4.50)
% r = risk-free yield (0.02)
% v = volatility; must be >0 (0.15)
% N = height of the binomial tree (20)
% OUTPUT:
% W = price of the compound Put option
% P = price of the vanilla European Put option
% EXAMPLE:
% [W,P] = CRRcp(1.05, 90, 95, 4.50, 0.02, 0.15, 20);
% W(1,1),P(1,1) % to get just W(0) and P(0)
% [pu,up,R]=CRRparams(T, r, v, N); % Use CRR values
% [C,P]=CRReur(T, S0, r, v, N); % vanilla options at T
% M=round(T1*N/T); % number of time steps to time T1
% W=zeros(M+1,M+1); % smaller output matrix
% for j=0:M % Set terminal values at expiry T1
% W(M+1,j+1) = max(P(M+1,j+1)−L, 0); % plus part
% end % ...for a Call option on a vanilla Put
% for n=M−1:-1:0 % Backward induction
% for j=0:n % Binomial pricing model value
% W(n+1,j+1)=(pu*W(n+2,j+2)+(1−pu)*W(n+2,j+1))/R;
% end
end
```
return % Prices in matrices W and P are defined.
end

Compute the compound Put premium with the requested parameters as follows:

\[ [W,P] = \text{CRRcp}(2,1,90,95,4.50,0.03,0.15,20); \]
\[ W(1,1), P(1,1) \% 4.0747, 7.3931 \]

Conclude that the premium is about \( 4.07 \).

2. Let \( \mathcal{M}(n) \) be the set of path numbers in a recombining binomial tree to depth \( n \), as defined in Eq.4.6. Prove that

\[ \mathcal{M}(n+1) = [2\mathcal{M}(n)] \cup [2\mathcal{M}(n)+1], \]

where \( aX + b \overset{\text{def}}{=} \{ax + b : x \in X\} \) for sets \( X \) of numbers.

**Solution:** Apply the definitions:

\[ \mathcal{M}(n) = \{2^n, 2^n+1, \ldots, 2^n + (2^n-1)\}, \]
\[ 2\mathcal{M}(n) = \{2^{n+1}, 2^{n+1}+2, \ldots, 2^{n+1} + (2^{n+1}-2)\}, \]
\[ 2\mathcal{M}(n) + 1 = \{2^{n+1} + 1, 2^{n+1}+3, \ldots, 2^{n+1} + (2^{n+1}-1)\}, \]

so the union of the last two sets is evidently

\[ \mathcal{M}(n+1) = \{2^{n+1}, 2^{n+1}+1, \ldots, 2^{n+1} + (2^{n+1}-1)\}, \]

as claimed.

3. Implement floating strike option pricing in the CRR model using geometric means instead of arithmetic means, as in \text{CRRgro} versus \text{CRRaro}. Compare the results on the suggested example inputs.

**Solution:** The Octave function \text{CRRflg()} differs from \text{CRRflt()} at lines 21 and 24, which are directly copied from lines 19 and 22 of \text{CRRgro()} where \( lU \) is computed and used. It also differs in the use of \text{Geom} instead of \text{Avg} and in a few updated comments.
% C0 = Call option premium at t=0.
% P0 = Put option premium at t=0.
% EXAMPLE:
% [C, P] = CRRflg(1, 90, 0.02, 0.20, 4);
% To get just the premiums at t=0, use
% C(1), P(1)
% [pu, up, R] = CRRparams(T, r, v, N);
Sbar=NRTCRR(S0, up,1/up,N); % expanded S tree
lU=NRTpsums(log(Sbar),N); % partial sums of logs
C=zeros(size(Sbar)); P=zeros(size(Sbar));
for m=2^N:(2∗2^N−1) % mindices at expiry
Geom = exp(lU(m)/(N+1)); % geometric mean
C(m) = max(0, Sbar(m)−Geom); % Call payoff
P(m) = max(0,Geom−Sbar(m)); % Put payoff
end % ... prices set at expiry.
for m=(2^N−1):−1:1 % recursive previous indices
C(m) = (pu∗C(2∗m+1) + (1−pu)∗C(2∗m))/R;
P(m) = (pu∗P(2∗m+1) + (1−pu)∗P(2∗m))/R;
end % ... all prices set by backward induction.
return
end

Compare (and test for bugs) by running the geometric mean and arithmetic mean functions on the same suggested example inputs:

[Cg,Pg]=CRRflg(1,90,0.02,0.20,4); Cg(1),Pg(1) % 4.6711, 3.4209
[Ca,Pa]=CRRflt(1,90,0.02,0.20,4); Ca(1),Pa(1) % 4.4853, 3.5920

The Call and Put premiums are similar, though not identical.

4. Use CRRaro to compute the average-rate Call and Put premiums in the CRR model for $S_0 = K = 100$, $T = 1$, $r = 0.05$, $v = 0.15$, and different values of $N$. Compare $C(0) - P(0)$ with the limit value in Eq.4.20.

Solution: The Octave commands below perform the computations with $N = 3, 5, 7, 9, 11$:

```octave
r=0.05; T=1; K=100; S0=100; v=0.15;
limit=S0*(1-exp(-r*T))/(r*T)-K/exp(r*T) % limit = 2.4182
[C,P]=CRRaro(T,S0,K,r,v,3); N3=C(1)-P(1) % N3 = 2.4250
[C,P]=CRRaro(T,S0,K,r,v,5); N5=C(1)-P(1) % N5 = 2.4223
[C,P]=CRRaro(T,S0,K,r,v,7); N7=C(1)-P(1) % N7 = 2.4211
[C,P]=CRRaro(T,S0,K,r,v,9); N9=C(1)-P(1) % N9 = 2.4205
[C,P]=CRRaro(T,S0,K,r,v,11); N11=C(1)-P(1) % N11 = 2.4201
```

The outputs suggest that every $N \geq 5$ achieves an approximation within one cent.
5. Implement floating strike option pricing in the CRR model using path-dependent Arrow-Debreu securities. Check that the results agree with CRRflt.

**Solution:** The Octave function `CRRfltAD()` differs from `CRRflt()` in three ways, just as `CRRaroAD()` differs from `CRRaro()`:

```octave
function [C0, P0] = CRRfltAD(T, S0, r, v, N)
% Octave/MATLAB function to price floating strike
% Call and Put options using path-dependent
% Arrow–Debreu expansions with the Cox–Ross–Rubinstein (CRR) binomial pricing model.
% INPUTS: (Example)
% T = expiration time in years (1)
% S0 = spot stock price (90)
% r = riskless yield per year (0.02)
% v = volatility; must be >0 (0.20)
% N = height of the tree (4)
% OUTPUTS:
% C0 = Call option premium at t=0.
% P0 = Put option premium at t=0.
% EXAMPLE:
% [C0, P0]=CRRfltAD(1, 90, 0.02, 0.20, 4);
% [pu, up, R]=CRRparams(T, r, v, N);
% Sbar=NRTICRR(S0, up, 1/up, N); % expanded S tree
% Ubar=NRTpsums(Sbar, N); % all partial sums
% Lbar=PathAD(pu*ones(N,N), R*ones(N,N), N);
% mN=2^N:(2*2^N−1); % all m−indices at expiry
% AvgN=Ubar(mN)/(N+1); % S averages at expiry
% C0=max(0, Sbar(mN)−AvgN)*Lbar(mN)'; % Call
% P0=max(0, AvgN−Sbar(mN))*Lbar(mN)'; % Put
% return % . inner products give A−D expansions
end
```

Compare (and test for bugs) by running both functions on the same suggested example inputs:

```octave
[C0, P0]=CRRfltAD(1, 90, 0.02, 0.20, 4)
[C, P]=CRRflt(1, 90, 0.02, 0.20, 4); C(1), P(1)
```

The outputs agree: \( C0=4.4853 = C(1) \) and \( P0=3.5920 = P(1) \).

6. Write an Octave program to compute the maximums along all paths in a non-recombining binary tree of depth \( N \). (Hint: modify \( NRTmin() \).)

**Solution:** Let \( \bar{S}_{max}(m) \) be the maximum value of \( S(n,j) \) along the path indexed by \( m \). It may be computed from the non-recombining tree \( \bar{S} \) by...
the following recursion:

\[
\begin{align*}
S_{\text{max}}(1) & \overset{\text{def}}{=} S(1), \\
\bar{S}_{\text{max}}(2m) & = \max\{\bar{S}_{\text{max}}(m), \bar{S}(2m)\}, \\
\bar{S}_{\text{max}}(2m+1) & = \max\{\bar{S}_{\text{max}}(m), \bar{S}(2m+1)\}.
\end{align*}
\]

Eq. 1 is easily implemented in Octave. Following the hint, start with \texttt{NRTmin()} and modify lines 15 and 16, replacing \texttt{min} with \texttt{max}.

```octave
function Maxb = NRTmax(Sbar, N)
% Octave/MATLAB function to compute maximums along paths in a non-recombining tree (NRT).
% INPUTS: (Example)
% Sbar = NRT array of length 2*2^N-1 (1:15)
% N = tree depth, must be >=0 (3)
% OUTPUT:
% Maxb = NRT of maximums, same size as Sbar
% EXAMPLE:
% Maxb = NRTmax(1:15,3)
% %
% Maxb=zeros(size(Sbar)); % allocate the output
% Maxb(1)=Sbar(1); % max along the trivial path
% for m=1:2^N-1 % all future times up to N-1
%     Maxb(2*m)=max(Maxb(m),Sbar(2*m)); % down
%     Maxb(2*m+1)=max(Maxb(m),Sbar(2*m+1)); % up
% end
return end
```

7. Implement lookback option pricing in the CRR model using path-dependent Arrow-Debreu securities. Check that the results agree with \texttt{CRRlb}. 

**Solution:** The Octave function \texttt{CRRlbAD()} differs from \texttt{CRRlb()} in three ways, just as \texttt{CRRaroAD()} differs from \texttt{CRRaro()}:

```octave
function [C0, P0] = CRRlbAD(T, S0, r, v, N)
% Octave/MATLAB function to price Lookback Call and Put options using path-dependent Arrow-Debreu expansions with the Cox-Ross-Rubinstein (CRR) binomial pricing model.
% INPUTS: (Example)
% T = expiration time in years (1)
% S0 = spot stock price (100)
% r = riskless yield per year (0.05)
% v = volatility; must be >0 (0.20)
% N = height of the tree (4)
% OUTPUTS:
% C0 = Call option premium at t=0
```
% P0 = Put option premium at t=0
% EXAMPLE:
% [C0,P0]=CRRlb(1,100,0.05,0.20,4)

[pU,up,R]=CRRparams(T,r,v,N);
Sbar=NRTCR(S0,up,1/up,N); % expanded S tree
MinS=NRTmin(Sbar,N); MaxS=NRTmax(Sbar,N);
Lbar=PathAD(pU*ones(N,N),R*ones(N,N),N);
mN=2^N:(2*2^N−1); % all m−indices at expiry
C0=(Sbar(mN)−MinS(mN))∗Lbar(mN)'; % Call
P0=(MaxS(mN)−Sbar(mN))∗Lbar(mN)'; % Put
return % ..inner products give A−D expansions

end

Compare (and test for bugs) by running both functions on the same suggested example inputs:

[C0,P0]=CRRlbAD(1,100,0.05,0.20,4)
[C,P]=CRRlb(1,100,0.05,0.20,4); C(1),P(1)

The outputs agree: C0=13.758=C(1) and P0=9.6589=P(1).

8. Implement ladder Put option pricing in the CRR model by modifying CRRladC appropriately. Check that the premium is at least as great as that for the vanilla European-style Put with the same parameters.

Solution: For the ladder Put option, there is a specified strike price $K$ and a decreasing ladder $\{L_i\}$ of in-the-money prices below $K$:

$$S(0) ≥ \min\{S(0), K\} > L_1 > \cdots > L_k.$$  

Let $M(T) = \min\{S(t) : 0 ≤ t ≤ T\}$ be the minimum price of $S$ up to expiry. Then the payoff is

$$P(T) \overset{\text{def}}{=} \max\{[K − S(T)]^+, K − L_i\}, \quad (2)$$

where $L_i$ is the lowest rung reached by $S(t)$ as recorded by $M(T)$:

$$L_{i+1} < M(T) ≤ L_i.$$  

In the discrete CRR implementation with $N$ time steps to expiry, both $S$ and $M$ are stored as linear arrays with $m$ indexing. The paths of length $N$ are indexed by $m \in M(N) = \{2^N, \ldots, 2^N + (2^N−1)\}$, so those are the indices in the non-recombining tree $P$ where terminal values are set using Eq.2. Then the option premium is found by backward induction as usual:

$$P(m) = \frac{pP(2m+1) + (1−p)P(2m)}{R}, \quad m = 2^N−1, 2^N−2, \ldots, 2, 1,$$

where $p$ is the risk neutral up probability from the CRR model, and $R$ is the riskless return per time step.
function LadP = CRRladP(T, S0, K, L, r, v, N)
% Octave/MATLAB function to price a ladder Put
% option using a Cox–Ross–Rubinstein (CRR) model.
% INPUTS: (Example)
% T = expiration time in years (1)
% S0 = spot price (50)
% K = strike price (55)
% L = decreasing prices < S0,K ([45,40,35])
% r = riskless yield per year (0.05)
% v = volatility; must be >0 (0.20)
% N = height of the tree (4)
% OUTPUTS:
% LadP = Ladder Put option price array.
% EXAMPLE:
% LadP = CRRladP(1,50,55,[45,40,35],0.05,0.20,4)
% [pu,up,R] = CRRparams(T,r,v,N);
% Sbar=NRTLRR(S0,up,1/up,N); % S as an NRT
MinS=NRTmin(Sbar,N); % non-recombining min tree
LadP=zeros(size(Sbar)); % Put prices NRT
% Initialize with the payoffs at expiry:
k=length(L); % number of ladder levels , >1
for m=2^N:2^N-1 % state indexes at expiry
    if (MinS(m)>L(1)) % above level L(1) is special
        LadP(m)=max(K-Sbar(m),0);
    else % MinS =< L(1)
        if (MinS(m)>L(k)) % ... use levels L(2)>...>L(k)
            for l=2:k % loop to find the level
                if (MinS(m)>L(1))
                    LadP(m)=max(max(K-Sbar(m),K-L(1-1)),0);
                    break; % found 1: L(1)<MinS=<L(1-1),
                end
            end % ... so exit the 'l' loop
        else % MinS=<L(k), another special case
            LadP(m)=max(max(K-Sbar(m),K-L(k)),0);
        end
    for m=(2^N-1):-1:1 % backward recursion
        LadP(m)=(pu*LadP(2*m+1)+(1-pu)*LadP(2*m))/R;
    end
    return % LadP(1) is the option premium
end

Note that, as with the Call,

$$P(T) \geq [K - S(T)]^+ \geq 0,$$

so this exotic option cannot cost less than a vanilla European Put with the same parameters. A few experiments will show how big the differences can be, for example,
LadP=CRRladrP(1,50,55,[45,40,35],0.05,0.20,4);LadP(1)  % 5.8784
[eC,eP]=CRReur(1,50,55,0.05,0.20,4);eP(1,1)  % 5.5372

The returned values round to $5.88 for the ladder Put versus $5.54 for the vanilla European Put, ordered as expected.