1. (a) We can formulate the problem as follows:

\[
\min \| y(\sigma) - y_0 \| \quad (0.1)
\]

where \( \sigma \) is a scalar, \( y_0 \) is a given vector of option prices, and \( y(\sigma) \) is a vector of option prices when the volatilities are \( \sigma \). Here we assume the volatility is constant.

The following are the Matlab code for \( y(\sigma) \) and the result.

```matlab
function y = fun1(s)
y0 = [80.0000
     60.0000
     40.0000
     22.5919
     10.8488
     2.3954
     0.3352
     0.1076
     0.0502
     0.0208
     0.000001];
S = [0:20:200]';
sig = s * ones(11, 1);X = 80; r = .1; T = 1; Q = 0;
[call, put] = blspricefdamsig(S, X, r, T, sig, Q);
prices = put(:, 20);
y = y0 - prices;
```

```matlab
>> [x, options, f, j] = leastsq('fun1', .2);
>> x
x =
  0.3929
>> norm(f)
an =
  3.8895
```

(b) Now \( y(\sigma, ir) \) is a function of the volatility and the interest rate.

```matlab
function y = fun2(s)
y0 = [80.0000
     60.0000
     40.0000
     22.5919
     10.8488
     2.3954
     0.3352
     0.1076
     0.0502
     0.0208
     0.000001];
S = [0:20:200]';
sig = s * ones(11, 1);X = 80; r = .1; T = 1; Q = 0;
[call, put] = blspricefdamsig(S, X, r, T, sig, Q);
prices = put(:, 20);
y = y0 - prices;
```

```matlab
>> [x, options, f, j] = leastsq('fun2', .2);
>> x
x =
  0.3929
>> norm(f)
an =
  3.8895
```
0.0208
0.000001];
S = [0:20:200]';
sig = s(1) * ones(11, 1);
X = 80; r = s(2); T = 1; Q = 0;
[call, put] = blspricefdamsig(S, X, r, T, sig, Q);
prices = put(:, 20);
y = y0 - prices;

-----------------------------------------------------------------------------------------------------------------
>> [x, options, f, j] = leastsq('fun2', [.3 .3]);
>> x
x =
  0.3193   -0.0035
>> norm(f)
an =
   2.3813

-----------------------------------------------------------------------------------------------------------------
The best volatility and interest rate are .3193 and -.0035 repectively. It doesn’t make a financial sense to have a negative interest rate, but leastsq can’t handle a box constraint.

2. The formulation of the problem is exactly the same as 1(a) except that we have a vector of volatilities corresponding to a option price vector instead of a constant volatility.

-----------------------------------------------------------------------------------------------------------------
function y = fun3(s)
y0 = [ 80.0000
     60.0000
     40.0000
     22.5919
     10.8488
     2.3954
     0.3352
     0.1076
     0.0502
     0.0208
     0.000001];
S = [0:20:200]';
sig = s;
X = 80; r = .1; T = 1; Q = 0;
[call, put] = blspricefdamsig(S, X, r, T, sig, Q);
prices = put(:, 20);
y = y0 - prices;

-----------------------------------------------------------------------------------------------------------------
>> [x, options, f, j] = leastsq('fun3', .3*ones(11,1));
>> x'
ans =
   Columns 1 through 7
   0.3000    0.3006    0.6019    0.5498    0.2500    0.2000    0.3000
   Columns 8 through 11
   0.4497    0.5006    0.3000    0.3000
>> norm(f)
an =
   1.8119e-06

-----------------------------------------------------------------------------------------------------------------
We can see that the residual is much smaller than the part (a) of problem 1.
3. The code which generates the figure is

```matlab
clear;

[cally, pusty, bdata, u] = blspricefdam(300, 100, .1, 1, .1, 0, 200, 300);
plot(bdata(:,1), bdata(:,2));
hold on;
for s = .2:.1:.9
    [call, put, bdata, u] = blspricefdam(300, 100, .1, 1, s, 0, 200, 300);
    plot(bdata(:,1), bdata(:,2)); drawnow;
end
axis([0 200 0 1]);
```

In the figure, the volatility is increasing from left to right. It makes sense since if the volatility is high, then there is more chance that stock price goes down and the option ends in the money. In that case, we have smaller region for exercising the option, which moves the free boundary to the left. If the volatility is very low, there is less chance that we gain more by waiting to exercise. It’s better to exercise the option early and to invest the cash flow in the money market account. That’s what moves the boundary to the right.

4. The modified binprice is

```matlab
function optprice = mbinprice(so,x,r,t,dt,sig,flag,q,div,exdiv)
%MBINPRICE Modified binomial put and call pricing.
% OPTPRICE = MBINPRICE(SO,X,R,T,DT,SIG,FLAG,Q,DIV,EXDIV)
% calculates a current option price using a binomial pricing model.
% SO is the underlying
% asset price, X is the option exercise price, R is the risk-free
% interest rate, T is the options time until maturity in years, DT
% is the time increment within T, SIG is the assets volatility, FLAG
% specifies whether the option is a call (flag = 1) or a put (flag = 0),
% Q is the dividend rate, DIV is the dividend payment at an ex-dividend
% date, EXDIV. EXDIV is specified in number of periods. All inputs to
```
this function are scalar values except DIV and EXDIV which are 1-by-n vectors. For each dividend payment, there must be a corresponding ex-dividend date. By default q, div, and EXDIV equal 0. If a value is entered for the dividend rate q, DIV and EXDIV should equal 0 or not be entered. If values are entered for DIV and EXDIV, set Q = 0.

See also BLSPRICE, BINPRICE.


if nargin < 8
    q = 0;
end
if nargin < 9
    div = 0;
    exdiv = 0;
end
if nargin < 7
    error(sprintf('Missing one of SO, X, R, T, DT, SIG, and FLAG.'))
end
if flag ~= 0 & flag ~= 1
    error(sprintf('FLAG must be 1 for call option or 0 put option.'))
end
if q ~= 0 & exist('div')
     if div ~= 0
        disp(setstr(7))
        fprintf(['??? Error using ==> mbinprice
', ...
        'DIV and EXDIV must be zero for non-zero dividend rate, Q.

'])
        return
    end
end

% Calculate the probability of an upward price movement
u = exp(sig.*sqrt(dt));
a = exp((r-q).*dt);
d = 1./u;
p = (a-d)./(u-d);

nper = round(t/dt);             % Number of periods after time zero
npp = nper+1;                   % Number of periods including time zero
pvdiv = div.*exp(-exdiv.*dt.*r); % Find present value of all dividends
so = so-sum(pvdiv(:));         % Find current price - div present values

if exist('div')                 % Present value of future dividends at nodes
    lendiv = length(div(:));
    lenexdiv = length(exdiv(:));
    if lendiv ~= lenexdiv
        error(sprintf('Number of dividend and ex-dividend entries must be equal.'))
    end
end

% Asset price at nodes at the maturity
pr = so*u.^[nper:-2:-nper] + ...
    sum((exdiv>(nper-1)).*div.*exp(-(exdiv-(nper-1)*dt)*r));
opt = zeros(size(pr));
if flag == 1  % Option is a call
    opt = max(pr-x,0);  % Determine option values from prices
    for n = nper:-1:1
        k = 1:n;
        discopt = (p*opt(k)+(1-p)*opt(k+1))*exp(-r*dt);
        pr = so*u.^[n-1:-2:1-n]’ + ... % Probable option values discounted back one time step
                sum((exdiv>(n-1)).*div.*exp(-(exdiv-(n-1))*dt*r)); % Option value is max of current price - X or discopt
        opt = [max(pr-x,discopt);zeros(npp-n,1)];
    end
elseif flag == 0  % Option is a put
    opt = max(x-pr,0);  % Determine option values from prices
    for n = nper:-1:1
        k = 1:n;
        discopt = (p*opt(k)+(1-p)*opt(k+1))*exp(-r*dt);
        pr = so*u.^[n-1:-2:1-n]’ + ... % Probable option values discounted back one time step
                sum((exdiv>(n-1)).*div.*exp(-(exdiv-(n-1))*dt*r)); % Option value is max of X - current price or discopt
        opt = [max(x-pr,discopt);zeros(npp-n,1)];
    end
end

optprice = opt(1);% Option price

The example we are investigating here is slightly different from the problem and is an American call option with no dividend whose exact value can be obtained via a Matlab financial toolbox function blsprice since an American call option and an European call option have the same value when there is no dividend and other parameters are the same. The underlying stock price is 31, the exercise price 40, the interest rate .1, the volatility .3, and the option price obtained by blsprice is 1.7676.

The following figure shows that there is some reduction in time, too. The elapsed time in the figure is measured by Matlab functions tic and toc.
The steep increase of time in \texttt{binprice} for large $M$ is due to the time needed for handling swap memory.

We compare the flops of both cases in the following figure. The flops are measured by a Matlab function \texttt{flops}.

We can see that \texttt{mbinprice} takes almost half of the flops as \texttt{binprice}.

One interesting observation is the convergence rate of both functions. \texttt{Binprice} seems to be very slow in convergence and doesn’t give an accurate answer for relatively large $M$, for example, around 1,000. On the other hand, \texttt{mbinprice} gives better results and shows regular convergence pattern in this particular case.

5. The following Matlab code generates the graph below.

```matlab
clear;
s = 100; x = 120; r = .01; t = 1; sig = .3; q = 0;
c = blsprice(s, x, r, t, sig, q);
dT = [.1 .05 .025 .01 .005 .0025 .001 .0005];
i = 1;
```
for dt = dT
    [pr, opt] = binprice(s, x, r, t, dt, sig, 1, q);
    c1(i) = opt(1,1); i = i + 1;
end

M = 100:100:500;

i = 1;
for m = M
    [call, put] = blspricefdam(s, x, r, t, sig, q, m, m);
    c2(i) = call; i = i + 1;
end

loglog(1./dT, abs(c1 - c)); hold on;
loglog(M, abs(c2 - c), '--');
axis([min(1./dT) max(1./dT) -inf inf]);

Both of binprice and blspricefdam converge to the right solution as the number of time steps increases. blspricefdam is doing slightly better than binprice for the same number of time steps even though the computational cost might be different.

6. The code

clear;

% part (a)

S = 100; X = 50; R = .1; T = 1; dT = .01; flag = 0; Q = 0; SIG = .3;
Smax = 200; dS = .5;

t = cputime; flops(0);
[pr, opt] = binprice(S, X, R, T, dT, SIG, flag, Q);
pbin = opt(1,1)
tbin = cputime - t
fbin = flops
\[ S, T \] = \text{meshgrid}(0:.1:200, 0:.05:1);
\[
t = \text{cputime}; \text{flops}(0);
\]
\[
[\text{call, put}] = \text{blspricefdam}(S, X, R, T, \text{SIG}, Q, 200, 200);
\]
\% \text{pfd} = \text{put}(1, \ S/\text{d}S+1)
\[
tfd = \text{cputime} - t
\]
\[
\text{ffd} = \text{flops}
\]

\begin{verbatim}
>> no6test2
pbin =
    0.0264
tbin =
    0.0900
fbin =
    97285
tfd =
    55.3200
ffd =
    4680283
>> \% part (a)
>> put(21, 1001)
ans =
    0.0268
>> \% part (b)
>> tbin = 2000 * tbin
tbin =
    180.0000
>> fbin = 2000 * fbin
fbin =
    194570000
>> \% part (c)
>> tbin = 19 * tbin
tbin =
    3.4200e+03
>> fbin = 19 * fbin
fbin =
    3.6968e+09
\end{verbatim}

blspricefdam computes all the option prices on the grid at once. In part (a),
blspricefdam seems to be inefficient for one underlying stock price, but it’s superior to
\text{binprice} in parts (b) and (c).

7. When \( S = 0 \), both of American and European vanilla put options have the same value, \( K \). At
\( t = 0 \), on the other hand, European option value is less than American option value, \( K \). Even
though it’s sure to get \( K \) dollars at the maturity with a European option, we should discount it
to get the present value of a European option.

8. American options have more value than European options since they give the right to exercise
before the maturity in addition to the right to exercise at the maturity.
9. (a)

The value of American option is greater than the payoff. The value of European option is never equal to B before the maturity since it is the discounted value of the payoff at the maturity.

(b)

The codes

```
function call = conamcall(Smax, dS, T, dT, X, B, R, SIG);
% Option price surface can be obtained via
%    mesh(0:dS:Smax, 0:dT:T, put);

% Smax : maximum stock price
% dS : increment of stock price
% T  : maturity date
% dT : time step
% X  : exercise price
% B  : payoff
% R  : risk free interest rate
% reference : John C. Hull, Options, Futures, and Other Derivatives
%            3rd Ed., Chap 15

M = ceil(Smax/dS); ds = Smax / M;
```
\[ N = \text{ceil}(T/dT); \quad dt = T / N; \]

\[ J = 1:M-1; \]
\[ a = 0.5R*dt*J - 0.5\text{SIG}^2*dt*J.^2; \]
\[ b = 1 + \text{SIG}^2*dt*J.^2 + R*dt; \]
\[ c = -0.5R*dt*J - 0.5\text{SIG}^2*dt*J.^2; \]

\[ A = \text{diag}(b) + \text{diag}(a(2:M-1), -1) + \text{diag}(c(1:M-2), 1); \]

\[ \text{call} = \text{zeros}(N+1, M+1); \]
\[ \text{call}(N+1, :) = B \times ([0:dS:Smax] >= X); \]
\[ \text{call}(:, M+1) = B; \]

\[ \text{for } i = N:-1:1 \]
\[ \quad y = \text{call}(i+1, 2:M)'; \quad y(M-1) = y(M-1) - c(M-1)\times\text{call}(i,M+1); \]
\[ \quad \text{call}(i, 2:M) = [A \setminus y]'; \]
\[ \quad \text{call}(i, :) = \max(B \times ([0:dS:Smax] >= X), \text{call}(i,:)); \]
\[ \text{end} \]

-----------------------------------------------------------------------------------------------------------------

\text{function } \text{call} = \text{coneucall}(Smax, dS, T, dT, X, B, R, SIG); \]

\[ M = \text{ceil}(Smax/dS); \quad ds = Smax / M; \]
\[ N = \text{ceil}(T/dT); \quad dt = T / N; \]

\[ J = 1:M-1; \]
\[ a = 0.5R*dt*J - 0.5\text{SIG}^2*dt*J.^2; \]
\[ b = 1 + \text{SIG}^2*dt*J.^2 + R*dt; \]
\[ c = -0.5R*dt*J - 0.5\text{SIG}^2*dt*J.^2; \]

\[ A = \text{diag}(b) + \text{diag}(a(2:M-1), -1) + \text{diag}(c(1:M-2), 1); \]

\[ \text{call} = \text{zeros}(N+1, M+1); \]
\[ \text{call}(N+1, :) = B \times ([0:dS:Smax] >= X); \]
\[ \% \text{call}(:, 1) = 0; \]
\[ \text{call}(:, M+1) = B \times \exp(R*dt*[-N:0]'); \]

\[ \text{for } i = N:-1:1 \]
\[ \quad y = \text{call}(i+1, 2:M)'; \quad y(M-1) = y(M-1) - c(M-1)\times\text{call}(i,M+1); \]
\[ \quad \text{call}(i, 2:M) = [A \setminus y]'; \]
\[ \% \text{put}(i, :) = \max(X - [0:ds:Smax], \text{put}(i,:)); \]
\[ \text{end} \]

-----------------------------------------------------------------------------------------------------------------