Question 1)
Code:

```
# importing the necessary packages
import numpy as np
from numpy import sqrt, exp, log
from scipy.stats import norm
import triSolver

################################################################################

# initial vars
S0 = 100.
sigma = 0.30
r = 0.02
T = 1.
K = 115.

# on interval [0, 200]
# assume Smax = 2*S0
Smax = 200.
ds = Smax/1000.      # space step
M = 1000
timestep = 100.  # change this according to question part
dt = T/timestep  # time step
N = int(T/dt)

# stock steps
s = np.array([ds*j for j in range(M)])

# initial v setup (put payoff)
v = np.maximum(K - s, 0)
vtemp = v

# basic alpha, beta, gamma matrix A version
alpha = 0.5  * (sigma**2*s**2/ds**2 - r*s/ds)
beta = -sigma**2*s**2/ds**2 - r
gamma = 0.5  * (sigma**2*s**2/ds**2 + r*s/ds)

# setting up matrix A
A = np.diag(beta) + np.diag(alpha[1:], -1) + np.diag(gamma[0:M-1], 1)

# boundary conditions for A
A[0, :] = 0
A[M-1, :] = 0
A[M-1, -3:] = np.array([1,-2,1])

# identity matrix
I = np.identity(1000)

# setting up main big matrix A_tilda for price equation
A_tilda = I - dt*A
#Ainv = np.linalg.inv(A_tilda)

# for price calculation purposes
exerciseTime = N - 0.75/dt + 1
incr = 1

# Bermudan option pricing calculation
for k in range(4):
```

```
    while (incr < (exerciseTime + k*25)):
        #vtemp = np.dot(Ainv, vtemp)
        vtemp = triSolver.betterSolver(A_tilda, vtemp)
        incr += 1

    v = np.maximum(v, vtemp)
    vtemp = v


# closed form BS formula price
d1 = ( log(S0/K) + (r + (sigma**2)/2)*T ) / ( sigma*sqrt(T) )
d2 = d1 - sigma*sqrt(T)
BSCallPrice =  K*exp(-r*T)*norm.cdf(-d2) - S0*norm.cdf(-d1)
print("Put Price explicitly via BS model formula = " + str(BSCallPrice))
print("-------------------------------------------------")

print("Implicit finite difference method estimated price (for timestep = " + str(timest
ep) + ") is:")
print(v[(M+1)/2])
```

Output:

Put Price explicitly via BS model formula = 20.0318695624

Implicit finite difference method estimated price (for timestep = 100.0) is:
20.3162757475

Question 2) a)

Code:

```
# initial vars
S0 = 100.
sigma = 0.30
r = 0.02
T = 1.
K = 115.

# on interval [0, 200]
# assume Smax = 2*S0
Smax = 200.
ds = Smax/1000.      # space step
M = 1000
timestep = 100.  # change this according to question part
dt = T/timestep  # time step
N = int(T/dt)

# stock steps
s = np.array([ds*j for j in range(M)])

# initial v setup (put payoff)
v = np.maximum(K - s, 0)

# basic alpha, beta, gamma matrix A version
alpha = 0.5  * (sigma**2*s**2/ds**2 - r*s/ds)
beta = -sigma**2*s**2/ds**2 - r
gamma = 0.5  * (sigma**2*s**2/ds**2 + r*s/ds)

# setting up matrix A
A = np.diag(beta) + np.diag(alpha[1:], -1) + np.diag(gamma[0:M-1], 1)

# boundary conditions for A
A[0, :] = 0
A[M-1, :] = 0
A[M-1, -3:] = np.array([1,-2,1])

# identity matrix
I = np.identity(1000)

# setting up main big matrix A_tilda for price equation
A_tilda = I - dt*A

# Bermudan approximation of American Put option pricing calculation
for k in range(N):
    vtemp = triSolver.betterSolver(A_tilda, v)
    v = np.maximum(v, vtemp)


# closed form BS formula price
d1 = ( log(S0/K) + (r + (sigma**2)/2)*T ) / ( sigma*sqrt(T) )
d2 = d1 - sigma*sqrt(T)
BSCallPrice =  K*exp(-r*T)*norm.cdf(-d2) - S0*norm.cdf(-d1)
print("Put Price explicitly via BS model formula = " + str(BSCallPrice))
print("-------------------------------------------------")

print("Implicit finite difference method estimated price (for timestep = " + str(timest
ep) + ") is:")
print(v[(M+1)/2])
```

Output:

Put Price explicitly via BS model formula = 20.0318695624

Implicit finite difference method estimated price (for timestep = 100.0) is:
20.4090231077

Question 2) b)

Code:

```
# initial vars
S0 = 100.
sigma = 0.30
r = 0.02
T = 1.
K = 115.

# on interval [0, 200]
# assume Smax = 2*S0
Smax = 200.
ds = Smax/1000.      # space step
M = 1000
timestep = 100.  # change this according to question part
dt = T/timestep  # time step
N = int(T/dt)

# stock steps
s = np.array([ds*j for j in range(M)])

# initial v setup (put payoff)
v = np.maximum(K - s, 0)
v_tilda = v

# basic alpha, beta, gamma matrix A version
alpha = 0.5  * (sigma**2*s**2/ds**2 - r*s/ds)
beta = -sigma**2*s**2/ds**2 - r
gamma = 0.5  * (sigma**2*s**2/ds**2 + r*s/ds)

# setting up matrix A
A = np.diag(beta) + np.diag(alpha[1:], -1) + np.diag(gamma[0:M-1], 1)

# boundary conditions for A
A[0, :] = 0
A[M-1, :] = 0
A[M-1, -3:] = np.array([1,-2,1])

# identity matrix
I = np.identity(1000)

# setting up main big matrix A_tilda for price equation
A_tilda = I - dt*A
#Ainv = np.linalg.inv(A_tilda)

# setting up matrix for Brennan-Schwartz algorithm
for i in range(0, len(A_tilda)-1):
    v_tilda[i+1] = v_tilda[i+1] - v_tilda[i]*A_tilda[i+1][i]/A_tilda[i][i]
    A_tilda[i+1] = A_tilda[i+1] - A_tilda[i]*A_tilda[i+1][i]/A_tilda[i][i]


# approximation of American Put option pricing calculation
```

```
51. for k in range(N):
52.     #vtemp = np.dot(Ainv,v)
53.     vtemp = triSolver.betterSolver(A_tilda, v_tilda)
54.     v = np.maximum(v, vtemp)
55.
56.
57. # closed form BS formula price
58. d1 = ( log(S0/K) + (r + (sigma**2)/2)*T ) / ( sigma*sqrt(T) )
59. d2 = d1 - sigma*sqrt(T)
60. BSCallPrice =  K*exp(-r*T)*norm.cdf(-d2) - S0*norm.cdf(-d1)
61. print("Put Price explicitly via BS model formula = " + str(BSCallPrice))
62. print("-------------------------------------------------")
63.
64. print("Implicit finite difference method estimated price (for timestep = " + str(timest
    ep) + ") is:")
65. print(v[(M+1)/2)
```

Output:

Put Price explicitly via BS model formula = 20.0318695624

Implicit finite difference method estimated price (for timestep = 100.0) is:
189.982548358

As we can see, the Bermudan approximate seems to be the result with the best accuracy.
The Bermudan approximation gives a result of ~20.
The Brennan-Schwartz algorithm gives a much larger price value.

Question 3)

Code:

```
# initial vars
S0 = 100.
sigma = 0.20
r = 0.03
T = 1.
K = 115.

# on interval [0, 200]
# assume Smax = 2*S0
Smax = 200.
ds = Smax/1000.      # space step
M = 1000
timestep = 100.  # change this according to question part
dt = T/timestep  # time step
N = int(T/dt)

# stock steps
s = np.array([ds*j for j in range(M)])

# initial v setup
# quadratic payoff
# power call
v = np.maximum(s - K, 0)
v = v**2

# basic alpha, beta, gamma matrix A version
alpha = 0.5  * (sigma**2*s**2/ds**2 - r*s/ds)
beta = -sigma**2*s**2/ds**2 - r
gamma = 0.5  * (sigma**2*s**2/ds**2 + r*s/ds)

# setting up matrix A
A = np.diag(beta) + np.diag(alpha[1:], -1) + np.diag(gamma[0:M-1], 1)

# boundary conditions for A
# only explicit boundary at 0
# special boundary condition at Smax last line
# (based on Prof's piazza post when he explained the previous linearity condition)
A[0, :] = 0

A[-1, -1] = r*Smax/ds + 0.5*sigma**2*Smax**2/(ds**2) - r
A[-1, -2] = -sigma**2*Smax**2/(ds**2) - r*Smax/ds
A[-1, -3] = 0.5*sigma**2*Smax**2/(ds**2)

# identity matrix
I = np.identity(1000)

# setting up LHS and RHS matrices for price equation
leftB = I - 0.5*dt*A
rightB = I + 0.5*dt*A

# Crank-Nicolson scheme
for k in range(N):
    b_tilda = np.dot(rightB, v)
    v = triSolver.betterSolver(leftB, b_tilda)

# closed form BS formula price
d1 = ( log(S0/K) + (r + (sigma**2)/2)*T ) / ( sigma*sqrt(T) )
d2 = d1 - sigma*sqrt(T)
```

```
59. BSCallPrice = S0*norm.cdf(d1) - K*exp(-r*T)*norm.cdf(d2)
60. print("Call Price explicitly via BS model formula = " + str(BSCallPrice))
61. print("-------------------------------------------------")
62.
63. print("Crank-
    Nicolson finite difference method estimated price (for timestep = " + str(timestep) + "
    ) is:")
64. print(v[(M+1)/2])
```

Output:

Call Price explicitly via BS model formula = 3.85958238114

Crank-Nicolson finite difference method estimated price (for timestep = 100.0) is:
106.985114029