

Question 1)

Code:

```
1. # importing the necessary packages
2. import numpy as np
3. from numpy import sqrt, exp, log
4. from scipy.stats import norm
5. import triSolver
6.
7. #####
8.
9. # initial vars
10. S0 = 100.
11. sigma = 0.30
12. r = 0.02
13. T = 1.
14. K = 115.
15.
16. # on interval [0, 200]
17. # assume Smax = 2*S0
18. Smax = 200.
19. ds = Smax/1000.      # space step
20. M = 1000
21. timestep = 100. # change this according to question part
22. dt = T/timestep # time step
23. N = int(T/dt)
24.
25. # stock steps
26. s = np.array([ds*j for j in range(M)])
27.
28. # initial v setup (put payoff)
29. v = np.maximum(K - s, 0)
30. vtemp = v
31.
32. # basic alpha, beta, gamma matrix A version
33. alpha = 0.5 * (sigma**2*s**2/ds**2 - r*s/ds)
34. beta = -sigma**2*s**2/ds**2 - r
35. gamma = 0.5 * (sigma**2*s**2/ds**2 + r*s/ds)
36.
37. # setting up matrix A
38. A = np.diag(beta) + np.diag(alpha[1:], -1) + np.diag(gamma[0:M-1], 1)
39.
40. # boundary conditions for A
41. A[0, :] = 0
42. A[M-1, :] = 0
43. A[M-1, -3:] = np.array([1, -2, 1])
44.
45. # identity matrix
46. I = np.identity(1000)
47.
48. # setting up main big matrix A_tilda for price equation
49. A_tilda = I - dt*A
50. #Ainv = np.linalg.inv(A_tilda)
51.
52. # for price calculation purposes
53. exerciseTime = N - 0.75/dt + 1
54. incr = 1
55.
56. # Bermudan option pricing calculation
57. for k in range(4):
```

```
58.     while (incr < (exerciseTime + k*25)):
59.         #vtemp = np.dot(Ainv, vtemp)
60.         vtemp = triSolver.betterSolver(A_tilda, vtemp)
61.         incr += 1
62.
63.     v = np.maximum(v, vtemp)
64.     vtemp = v
65.
66.
67. # closed form BS formula price
68. d1 = ( log(S0/K) + (r + (sigma**2)/2)*T ) / ( sigma*sqrt(T) )
69. d2 = d1 - sigma*sqrt(T)
70. BSCallPrice = K*exp(-r*T)*norm.cdf(-d2) - S0*norm.cdf(-d1)
71. print("Put Price explicitly via BS model formula = " + str(BSCallPrice))
72. print("-----")
73.
74. print("Implicit finite difference method estimated price (for timestep = " + str(timest
    ep) + ") is:")
75. 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:

```
1. # initial vars
2. S0 = 100.
3. sigma = 0.30
4. r = 0.02
5. T = 1.
6. K = 115.
7.
8. # on interval [0, 200]
9. # assume Smax = 2*S0
10. Smax = 200.
11. ds = Smax/1000.      # space step
12. M = 1000
13. timestep = 100. # change this according to question part
14. dt = T/timestep # time step
15. N = int(T/dt)
16.
17. # stock steps
18. s = np.array([ds*j for j in range(M)])
19.
20. # initial v setup (put payoff)
21. v = np.maximum(K - s, 0)
22.
23. # basic alpha, beta, gamma matrix A version
24. alpha = 0.5 * (sigma**2*s**2/ds**2 - r*s/ds)
25. beta = -sigma**2*s**2/ds**2 - r
26. gamma = 0.5 * (sigma**2*s**2/ds**2 + r*s/ds)
27.
28. # setting up matrix A
29. A = np.diag(beta) + np.diag(alpha[1:], -1) + np.diag(gamma[0:M-1], 1)
30.
31. # boundary conditions for A
32. A[0, :] = 0
33. A[M-1, :] = 0
34. A[M-1, -3:] = np.array([1,-2,1])
35.
36. # identity matrix
37. I = np.identity(1000)
38.
39. # setting up main big matrix A_tilda for price equation
40. A_tilda = I - dt*A
41.
42. # Bermudan approximation of American Put option pricing calculation
43. for k in range(N):
44.     vtemp = triSolver.betterSolver(A_tilda, v)
45.     v = np.maximum(v, vtemp)
46.
47.
48. # closed form BS formula price
49. d1 = ( log(S0/K) + (r + (sigma**2)/2)*T ) / ( sigma*sqrt(T) )
50. d2 = d1 - sigma*sqrt(T)
51. BSCallPrice = K*exp(-r*T)*norm.cdf(-d2) - S0*norm.cdf(-d1)
52. print("Put Price explicitly via BS model formula = " + str(BSCallPrice))
53. print("-----")
54.
55. print("Implicit finite difference method estimated price (for timestep = " + str(timestep) + ") is:")
56. 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:

```
1. # initial vars
2. S0 = 100.
3. sigma = 0.30
4. r = 0.02
5. T = 1.
6. K = 115.
7.
8. # on interval [0, 200]
9. # assume Smax = 2*S0
10. Smax = 200.
11. ds = Smax/1000.      # space step
12. M = 1000
13. timestep = 100. # change this according to question part
14. dt = T/timestep # time step
15. N = int(T/dt)
16.
17. # stock steps
18. s = np.array([ds*j for j in range(M)])
19.
20. # initial v setup (put payoff)
21. v = np.maximum(K - s, 0)
22. v_tilda = v
23.
24. # basic alpha, beta, gamma matrix A version
25. alpha = 0.5 * (sigma**2*s**2/ds**2 - r*s/ds)
26. beta = -sigma**2*s**2/ds**2 - r
27. gamma = 0.5 * (sigma**2*s**2/ds**2 + r*s/ds)
28.
29. # setting up matrix A
30. A = np.diag(beta) + np.diag(alpha[1:], -1) + np.diag(gamma[0:M-1], 1)
31.
32. # boundary conditions for A
33. A[0, :] = 0
34. A[M-1, :] = 0
35. A[M-1, -3:] = np.array([1, -2, 1])
36.
37. # identity matrix
38. I = np.identity(1000)
39.
40. # setting up main big matrix A_tilda for price equation
41. A_tilda = I - dt*A
42. #Ainv = np.linalg.inv(A_tilda)
43.
44. # setting up matrix for Brennan-Schwartz algorithm
45. for i in range(0, len(A_tilda)-1):
46.     v_tilda[i+1] = v_tilda[i+1] - v_tilda[i]*A_tilda[i+1][i]/A_tilda[i][i]
47.     A_tilda[i+1] = A_tilda[i+1] - A_tilda[i]*A_tilda[i+1][i]/A_tilda[i][i]
48.
49.
50. # 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:

```

1. # initial vars
2. S0 = 100.
3. sigma = 0.20
4. r = 0.03
5. T = 1.
6. K = 115.
7.
8. # on interval [0, 200]
9. # assume Smax = 2*S0
10. Smax = 200.
11. ds = Smax/1000.      # space step
12. M = 1000
13. timestep = 100.      # change this according to question part
14. dt = T/timestep      # time step
15. N = int(T/dt)
16.
17. # stock steps
18. s = np.array([ds*j for j in range(M)])
19.
20. # initial v setup
21. # quadratic payoff
22. # power call
23. v = np.maximum(s - K, 0)
24. v = v**2
25.
26. # basic alpha, beta, gamma matrix A version
27. alpha = 0.5 * (sigma**2*s**2/ds**2 - r*s/ds)
28. beta = -sigma**2*s**2/ds**2 - r
29. gamma = 0.5 * (sigma**2*s**2/ds**2 + r*s/ds)
30.
31. # setting up matrix A
32. A = np.diag(beta) + np.diag(alpha[1:], -1) + np.diag(gamma[0:M-1], 1)
33.
34. # boundary conditions for A
35. # only explicit boundary at 0
36. # special boundary condition at Smax last line
37. # (based on Prof's piazza post when he explained the previous linearity condition)
38. A[0, :] = 0
39.
40. A[-1, -1] = r*Smax/ds + 0.5*sigma**2*Smax**2/(ds**2) - r
41. A[-1, -2] = -sigma**2*Smax**2/(ds**2) - r*Smax/ds
42. A[-1, -3] = 0.5*sigma**2*Smax**2/(ds**2)
43.
44. # identity matrix
45. I = np.identity(1000)
46.
47. # setting up LHS and RHS matrices for price equation
48. leftB = I - 0.5*dt*A
49. rightB = I + 0.5*dt*A
50.
51. # Crank-Nicolson scheme
52. for k in range(N):
53.     b_tilda = np.dot(rightB, v)
54.     v = triSolver.betterSolver(leftB, b_tilda)
55.
56. # closed form BS formula price
57. d1 = ( log(S0/K) + (r + (sigma**2)/2)*T ) / ( sigma*sqrt(T) )
58. 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