```
 1: import numpy as np
 2: import matplotlib.pyplot as plt
 3:
 4: np.set_printoptions(precision=2)
 5:
 6:
 7: N = 10   # number of elements
 8: a = 1
 9: alpha = 1
10: g_b = 1
11: f_const = 1
12:
13: x = np.linspace(0, 1, N + 1)
14:
15:
16: A = np.zeros((N + 1, N + 1))
17: f_vec = np.zeros(N + 1)
18:
19: for i in range(1, N + 1):
20:     h = x[i] - x[i - 1]
21:
22:
23:     a_11 =  1./h + a*h/3.
24:     a_12 = -1./h + a*h/6.
25:     a_21 = -1./h + a*h/6.
26:     a_22 =  1./h + a*h/3
27:
28:     A[i - 1, i - 1] += a_11
29:     A[i - 1, i] += a_12
30:     A[i, i - 1] += a_21
31:     A[i, i] += a_22
32:
33:
34:     f_vec[i] = f_const*h
35:
36: print("A =\n", A)
37:
38: # take Neumann data into account
39: A[N, N] += alpha
40: f_vec[N] += alpha*g_b
41:
42:
43: # take Dirichlet data into account
44: u_g = np.zeros(N + 1)
45: u_g[0] = 0
46: print("u_g =\n", u_g)
47:
48: # remove first row of A
49: A_g = A[1:N+1, :]
50: #print("A_g =\n", A_g)
51:
52: # remove first row of f_vec
53: f_g = f_vec[1:N+1]
54:
55: # assemble RHS with dirichlet data
56: f_g -= A_g.dot(u_g)
57: #print(f_g)
58:
59: # matrix for the inner nodes (excluding nodes with dirichlet bcs)
60: A_0 = A[1:N+1, 1:N+1]
61: #print(A_0)
62:
63: # solve for u_0 (free dofs)
64: u_0 = np.linalg.solve(A_0, f_g)
65:
66: # assemble "u = u_0 + u_g"
67: u = np.concatenate([[0], u_0])
68:
```

```
69:
70: print("u =\n", u)
71:
72: plt.plot(x, u, '-')
73: plt.xlabel('x')
74: plt.ylabel('u_h(x)')
75: plt.grid()
76: plt.show()
```

```python
import numpy as np
import matplotlib.pyplot as plt

np.set_printoptions(precision=2)



N = 2   # number of elements
split = np.sqrt(2)/2
#split = 0.5


N_lower = round(N*split)            # number of elements in (0, split)
print(N_lower, "elements below sqrt(2)/2")
N_upper = N - N_lower               # number of elements in (split, 1)
print(N_upper, "elements above sqrt(2)/2")

assert(N == N_lower + N_upper)



x_lower = np.linspace(0, split, N_lower + 1)
x_upper = np.linspace(split, 1, N_upper + 1)
x = np.concatenate([x_lower, x_upper[1:]])
print(x)


A = np.zeros((N + 1, N + 1))

for i in range(1, N + 1):
    h = x[i] - x[i - 1]

    lam = 1
    if(x[i] > split):
        lam = 10


    a_11 = lam/h
    a_12 = -lam/h
    a_21 = -lam/h
    a_22 = lam/h

    A[i - 1, i - 1] += a_11
    A[i - 1, i] += a_12
    A[i, i - 1] += a_21
    A[i, i] += a_22

print("A =\n", A)



# take dirichlet data into account
u_g = np.zeros(N + 1)
u_g[0] = 0
u_g[N] = 1
print("u_g =\n", u_g)

# remove first and last row of A
A_g = A[1:N, :]
#print("A_g =\n", A_g)

# assemble RHS with dirichlet data
f = -A_g.dot(u_g)
#print(f)

# matrix for the inner nodes (excluding nodes with dirichlet bcs)
A_0 = A[1:N, 1:N]
#print(A_0)
```

```
69:
70: # solve for u_0 (free dofs)
71: u_0 = np.linalg.solve(A_0, f)
72:
73: # assemble "u = u_0 + u_g"
74: u = np.concatenate([[0], u_0, [1]])
75:
76:
77: print("u =\n", u)
78:
79: plt.plot(x, u, '-')
80: plt.xlabel('x')
81: plt.ylabel('u_h(x)')
82: plt.grid()
83: plt.show()
```

```
 1: import numpy as np
 2: import matplotlib.pyplot as plt
 3:
 4: np.set_printoptions(precision=2)
 5:
 6:
 7: p = 70
 8: N_vec = [10, 20, 30, 40, 70]
 9: for N in N_vec:
10:
11:     x = np.linspace(0, 1, N + 1)
12:
13:     A = np.zeros((N + 1, N + 1))
14:
15:     for i in range(1, N + 1):
16:         h = x[i] - x[i - 1]
17:
18:
19:         a_11 =  1./h - p/2.
20:         a_12 = -1./h + p/2.
21:         a_21 = -1./h - p/2.
22:         a_22 =  1./h + p/2.
23:
24:         A[i - 1, i - 1] += a_11
25:         A[i - 1, i] += a_12
26:         A[i, i - 1] += a_21
27:         A[i, i] += a_22
28:
29:     print("A =\n", A)
30:
31:
32:
33:     # take dirichlet data into account
34:     u_g = np.zeros(N + 1)
35:     u_g[0] = 0
36:     u_g[N] = 1
37:     print("u_g =\n", u_g)
38:
39:     # remove first and last row of A
40:     A_g = A[1:N, :]
41:     #print("A_g =\n", A_g)
42:
43:     # assemble RHS with dirichlet data
44:     f = -A_g.dot(u_g)
45:     #print(f)
46:
47:     # matrix for the inner nodes (excluding nodes with dirichlet bcs)
48:     A_0 = A[1:N, 1:N]
49:     #print(A_0)
50:
51:     # solve for u_0 (free dofs)
52:     u_0 = np.linalg.solve(A_0, f)
53:
54:     # assemble "u = u_0 + u_g"
55:     u = np.concatenate([[0], u_0, [1]])
56:
57:
58:     print("u =\n", u)
59:
60:     plt.plot(x, u, '-')
61:
62:
63: # plotting the exact solution
64: plt.plot(x, (np.exp(p*x) - 1.)/(np.exp(p) - 1.))
65:
66: plt.xlabel('x')
67: plt.ylabel('u_h(x)')
68: plt.legend(N_vec + ['exact'])
```

```
69: plt.title("Comparing discrete solution for increasing number of elements")
70: plt.grid()
71: plt.show()
72:
73:
74:
75:
76:
```