```python
import numpy as np

def flux_jumps(mesh, u):
    N = len(mesh) - 1   # number of elements
    jumps = np.zeros(N + 1) # N-1 edges

    for i in range(1, N):
        upper = (u[i + 1] - u[i])/(mesh[i + 1] - mesh[i])
        lower = (u[i] - u[i - 1])/(mesh[i] - mesh[i - 1])
        jumps[i] = upper - lower

    return jumps


def residual_errors(mesh, u):
    N = len(mesh) - 1   # number of elements
    errors = np.zeros(N)

    jumps = flux_jumps(mesh, u)

    for i in range(N):
        errors[i] = np.sqrt((jumps[i]**2 + jumps[i + 1]**2)/2)  # Braess (8.10)

    #print("errors:\n", errors)

    return errors



def adapt_h(mesh, u, alpha):
    N = len(mesh) - 1   # number of elements

    errors = residual_errors(mesh, u)
    threshhold = alpha * abs(max(errors))

    # refine mesh
    refined_mesh = [mesh[0]]
    for i in range(N):
        if abs(errors[i]) <= threshhold:
            refined_mesh.append(mesh[i + 1])
        else:
            refined_mesh.append(mesh[i] + (mesh[i + 1] - mesh[i])/2)
            refined_mesh.append(mesh[i + 1])


    #print("refined mesh:\n", refined_mesh)

    return refined_mesh


def adapt_r(mesh, u):
    N = len(mesh) - 1   # number of elements

    rho = np.abs(flux_jumps(mesh, u))  # rho ... mesh density function

    p = np.zeros(N)      # piecewise constant function on the mesh elements
    for i in range(N):
        p[i] = (rho[i] + rho[i + 1])/2

    P = np.zeros(N + 1)      # \int_0^{x_j} p(x) dx for j = 1,...,N
    for j in range(1, N + 1):
        h_j = mesh[j] - mesh[j - 1]
        P[j] = P[j - 1] + h_j*p[j - 1]  # add integral over j-th interval

    moved_mesh = np.zeros(N + 1)
    moved_mesh[0] = mesh[0]
    moved_mesh[-1] = mesh[-1]

```

```
    69:     for j in range(1, N):  # calculate the new nodes with De Boor's algorithm
    70:         xi_j = j/N
    71:         k = np.searchsorted(P, xi_j*P[-1], side="left")  # searches for index k, suc
h that xi_j*P[-1] > P[k]
    72:         assert(P[k - 1] < xi_j*P[-1])
    73:         assert(xi_j*P[-1] <= P[k])
    74:
    75:         moved_mesh[j] = mesh[k - 1] + (xi_j*P[-1] - P[k - 1])/p[k - 1]
    76:         print("orign_mesh[j] =", mesh[j])
    77:         print("moved_mesh[j] =", moved_mesh[j])
    78:     print("done\n")
    79:
    80:
    81:     #print("moved mesh:\n", moved_mesh)
    82:
    83:     return moved_mesh
```

```
    1: import numpy as np
    2: import scipy.integrate as integrate
    3: import matplotlib.pyplot as plt
    4: import adaptivity_schemes
    5:
    6: np.set_printoptions(precision=2)
    7:
    8:
    9: def Solve_6A(mesh, p):
   10:     N = len(mesh) - 1    # number of elements
   11:
   12:     f = lambda x : 2*p**3*x/((p**2*x**2 + 1)**2)
   13:     g_b = p/(p**2 + 1)
   14:
   15:
   16:
   17:     A = np.zeros((N + 1, N + 1))
   18:     f_vec = np.zeros(N + 1)
   19:
   20:     for i in range(1, N + 1):
   21:         h = mesh[i] - mesh[i - 1]
   22:
   23:         a_11 =  1./h
   24:         a_12 = -1./h
   25:         a_21 = -1./h
   26:         a_22 =  1./h
   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:         phi_lower = lambda x : (mesh[i] - x)/h
   35:         f_vec[i-1] += integrate.quad(lambda x : f(x)*phi_lower(x), mesh[i - 1], mesh
[i])[0]
   36:
   37:         phi_upper = lambda x : (x - mesh[i - 1])/h
   38:         f_vec[i] += integrate.quad(lambda x : f(x)*phi_upper(x), mesh[i - 1], mesh[i
])[0]
   39:
   40:
   41:
   42:     # take Neumann data into account
   43:     A[N, N] += 0
   44:     f_vec[N] += g_b
   45:
   46:
   47:     # take Dirichlet data into account
   48:     u_g = np.zeros(N + 1)
   49:     u_g[0] = -np.arctan(p)
   50:     #print("u_g =\n", u_g)
   51:
   52:     # remove first row of A
   53:     A_g = A[1:N+1, :]
   54:     #print("A_g =\n", A_g)
   55:
   56:     # remove first row of f_vec
   57:     f_g = f_vec[1:N+1]
   58:
   59:     # assemble RHS with dirichlet data
   60:     f_g -= A_g.dot(u_g)
   61:     #print("f_g =\n", f_g)
   62:
   63:     # matrix for the inner nodes (excluding nodes with dirichlet bcs)
   64:     A_0 = A[1:N+1, 1:N+1]
   65:     #print(A_0)
   66:
```

```
    67:     # solve for u_0 (free dofs)
    68:     u_0 = np.linalg.solve(A_0, f_g)
    69:
    70:     # assemble "u = u_0 + u_g"
    71:     u = np.concatenate([[u_g[0]], u_0])
    72:
    73:     return u
    74:
    75:
    76:
    77: p = 100
    78: ########## h-adaptivity ##########
    79: N = 5   # number of elements
    80: mesh = np.linspace(-1, 1, N + 1)
    81: u = Solve_6A(mesh, p)
    82:
    83: plt.plot(mesh, u, '-o')
    84: plt.grid()
    85: plt.xlabel('x')
    86: plt.ylabel('u_h(x)')
    87: plt.title("h-adaptivity")
    88:
    89:
    90: N_vec = ["0 refinements, " + str(N) + " elements"]
    91: refinements = 4   # number of refinements
    92: for i in range(refinements):
    93:     mesh = adaptivity_schemes.adapt_h(mesh, u, 0.7)
    94:     u = Solve_6A(mesh, p)
    95:     plt.plot(mesh, u, '-o')
    96:
    97:     N_vec.append(str(i + 1) + " refinements, " + str(len(mesh) - 1) + " elements")
    98:
    99:
   100: # plot exact solution
   101: x = np.linspace(-1, 1, 50)
   102: plt.plot(x, np.arctan(p*x))
   103: N_vec.append("exact")
   104:
   105: plt.legend(N_vec)
   106: plt.show()
   107:
   108:
   109:
   110: # ########## r-adaptivity ##########
   111: N = 5
   112: mesh = np.linspace(-1, 1, N + 1)
   113: u = Solve_6A(mesh, p)
   114: plt.plot(mesh, u, '-o')
   115: title = "r-adaptivity with " + str(N) + " elements"
   116: plt.title(title)
   117:
   118: adaptations_vec = ["0 adaptations"]
   119: adaptations = 5   # number of iterations
   120: for i in range(adaptations):
   121:     mesh = adaptivity_schemes.adapt_r(mesh, u)
   122:
   123:     u = Solve_6A(mesh, p)
   124:     plt.plot(mesh, u, '-o')
   125:
   126:     adaptations_vec.append(str(i + 1) + " adaptations")
   127:
   128:
   129: # plot exact solution
   130: x = np.linspace(-1, 1, 50)
   131: plt.plot(x, np.arctan(p*x))
   132: adaptations_vec.append("exact")
   133:
   134:
```

```
135: plt.legend(adaptations_vec)
136: plt.xlabel('x')
137: plt.ylabel('u_h(x)')
138: plt.grid()
139: plt.show()
```

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

np.set_printoptions(precision=2)


def lam_func(x):
    n = len(x)
    lam_vec = np.zeros(n)
    for i in range(n):
        if (x[i] > 1/np.sqrt(2)):
            lam_vec[i] = 10
        else:
            lam_vec[i] = 1
    return lam_vec



def Solve_6B(mesh):
    N = len(mesh) - 1    # number of elements

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

    lam_vec = lam_func(mesh)

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

        a_11 = lam_vec[i]/h
        a_12 = -lam_vec[i]/h
        a_21 = -lam_vec[i]/h
        a_22 = lam_vec[i]/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)

    # solve for u_0 (free dofs)
    u_0 = np.linalg.solve(A_0, f)

    # assemble "u = u_0 + u_g"
    u = np.concatenate([[0], u_0, [1]])
    #print("u =\n", u)

    return u

```

```
 69:
 70: ########## h-adaptivity ##########
 71: N = 2   # number of elements
 72: mesh = np.linspace(0, 1, N + 1)
 73: u = Solve_6B(mesh)
 74:
 75: plt.plot(mesh, u, '-o')
 76: plt.grid()
 77: plt.xlabel('x')
 78: plt.ylabel('u_h(x)')
 79: plt.title("h-adaptivity")
 80:
 81:
 82: N_vec = ["0 refinements, " + str(N) + " elements"]
 83: refinements = 5   # number of refinements
 84: for i in range(refinements):
 85:     mesh = adaptivity_schemes.adapt_h(mesh, lam_func(mesh)*u, 0.9)
 86:     u = Solve_6B(mesh)
 87:     plt.plot(mesh, u, '-o')
 88:
 89:     N_vec.append(str(i + 1) + " refinements, " + str(len(mesh) - 1) + " elements")
 90:
 91: plt.legend(N_vec)
 92: plt.show()
 93:
 94:
 95:
 96: ########## r-adaptivity ##########
 97: N = 5
 98: mesh = np.linspace(0, 1, N + 1)
 99: u = Solve_6B(mesh)
100: plt.plot(mesh, u, '-o')
101: title = "r-adaptivity with " + str(N) + " elements"
102: plt.title(title)
103:
104: adaptations_vec = ["0 adaptations"]
105: adaptations = 4   # number of iterations
106: for i in range(adaptations):
107:     mesh = adaptivity_schemes.adapt_r(mesh, lam_func(mesh)*u)
108:     u = Solve_6B(mesh)
109:     plt.plot(mesh, u, '-o')
110:
111:     adaptations_vec.append(str(i + 1) + " adaptations")
112:
113:
114: plt.legend(adaptations_vec)
115: plt.xlabel('x')
116: plt.ylabel('u_h(x)')
117: plt.grid()
118: plt.show()
119:
120:
```

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

np.set_printoptions(precision=2)


def Solve_6C(mesh, p):
    N = len(mesh) - 1   # number of elements

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

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

        a_11 =  1./h - p/2.
        a_12 = -1./h + p/2.
        a_21 = -1./h - p/2.
        a_22 =  1./h + p/2.

        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)

    # solve for u_0 (free dofs)
    u_0 = np.linalg.solve(A_0, f)

    # assemble "u = u_0 + u_g"
    u = np.concatenate([[0], u_0, [1]])

    return u









p = 70
######### h-adaptivity ##########
N = 5   # number of elements
mesh = np.linspace(0, 1, N + 1)
u = Solve_6C(mesh, p)

plt.plot(mesh, u, '-o')
plt.grid()
plt.xlabel('x')
```

```
 69: plt.ylabel('u_h(x)')
 70: plt.title("h-adaptivity")
 71:
 72:
 73: N_vec = ["0 refinements, " + str(N) + " elements"]
 74: refinements = 4   # number of refinements
 75: for i in range(refinements):
 76:     mesh = adaptivity_schemes.adapt_h(mesh, u, 0.7)
 77:     u = Solve_6C(mesh, p)
 78:     plt.plot(mesh, u, '-o')
 79:
 80:     N_vec.append(str(i + 1) + " refinements, " + str(len(mesh) - 1) + " elements")
 81:
 82:
 83: # plot exact solution
 84: x = np.linspace(0, 1, 50)
 85: plt.plot(x, (np.exp(p*x) - 1.)/(np.exp(p) - 1.))
 86: N_vec.append("exact")
 87:
 88: plt.legend(N_vec)
 89: plt.show()
 90:
 91:
 92:
 93: ########## r-adaptivity ##########
 94: N = 10
 95: mesh = np.linspace(0, 1, N + 1)
 96: u = Solve_6C(mesh, p)
 97: plt.plot(mesh, u, '-o')
 98: title = "r-adaptivity with " + str(N) + " elements"
 99: plt.title(title)
100:
101: adaptations_vec = ["0 adaptations"]
102: adaptations = 4   # number of iterations
103: for i in range(adaptations):
104:     mesh = adaptivity_schemes.adapt_r(mesh, u)
105:     u = Solve_6C(mesh, p)
106:     plt.plot(mesh, u, '-o')
107:
108:     adaptations_vec.append(str(i + 1) + " adaptations")
109:
110:
111: # plot exact solution
112: x = np.linspace(0, 1, 50)
113: plt.plot(x, (np.exp(p*x) - 1.)/(np.exp(p) - 1.))
114: adaptations_vec.append("exact")
115:
116:
117: plt.legend(adaptations_vec)
118: plt.xlabel('x')
119: plt.ylabel('u_h(x)')
120: plt.grid()
121: plt.show()
122:
123:
124:
125:
126:
127:
```