

Exercise 3

Part 1

Casey Allen

This notebook contains all numerical solutions to exercise 3, parts 1 and 2.

```
In [1]: from firedrake import *
import matplotlib.pyplot as plt
import numpy as np

def solve_poisson(nx=128, ny=None, cg_order=1, visualize=True, save_vtk=False):
    """
    Solve the Poisson equation using two methods and compare results.

    Parameters:
    -----
    nx      : Number of mesh cells in x-direction (default: 128)
    ny      : Number of mesh cells in y-direction (default: None, uses nx)
    cg_order : Order of continuous Galerkin elements (default: 1)
    visualize: Whether to create matplotlib plots (default: True)
    save_vtk : Whether to save VTK output for Paraview (default: True)

    Returns:
    -----
    dict : Dictionary containing solutions, errors, and mesh info
    """
    if ny is None:
        ny = nx

    # Create mesh
    mesh = UnitSquareMesh(nx, ny, quadrilateral=True)  #-----Creation of Mesh

    # Function space with specified CG order
    V = FunctionSpace(mesh, 'CG', cg_order)

    # Spatial coordinates and source term
    x, y = SpatialCoordinate(mesh)
    f = Function(V).interpolate(2*pi**2*sin(pi*x)*cos(pi*y))

    # Exact solution for error computation
    u_exact = Function(V).interpolate(sin(pi*x)*cos(pi*y))

    # Boundary conditions
    bc_x0 = DirichletBC(V, Constant(0), 1)
    bc_x1 = DirichletBC(V, Constant(0), 2)
    bcs = [bc_x0, bc_x1]

    # ===== Method 1: Weak form manually constructed - Step 1: Weak formulation
    u = TrialFunction(V)
    du = TestFunction(V)                      # Test function - equivalent to w(x,y)
    a = inner(grad(u), grad(du)) * dx #
    L = f * du * dx
```

```

u_1 = Function(V, name='Method_1_Manual')
solve(a == L, u_1, solver_parameters={'ksp_type': 'cg', 'pc_type': 'none'},

# ===== Method 2: Variational principle via derivative - Step 2: Ritz-Galerkin
u_2 = Function(V, name='Method_2_Variational')
Ju = (0.5*inner(grad(u_2), grad(u_2)) - u_2*f) * dx # Weak formulation - 1/2
F = derivative(Ju, u_2, du)                                # Build algebraic system
solve(F == 0, u_2, bcs=bcs)                               # Solve algebraic system

# ===== Error computation =====
L2_1 = sqrt(assemble(dot(u_1 - u_exact, u_1 - u_exact) * dx))
L2_2 = sqrt(assemble(dot(u_2 - u_exact, u_2 - u_exact) * dx))
L2_diff = sqrt(assemble(dot(u_2 - u_1, u_2 - u_1) * dx))

# Print results
print(f'\n{"*60}')
print(f'Mesh resolution: {nx}x{ny}, Δx = {1/nx:.6f}, CG order = {cg_order}')
print(f'{"*60}')
print(f'L2 error (Method 1 vs exact): {L2_1:.6e}')
print(f'L2 error (Method 2 vs exact): {L2_2:.6e}')
print(f'L2 norm (Method 2 - Method 1): {L2_diff:.6e}')
print(f'{"*60}\n')

# ===== Save VTK for Paraview =====
if save_vtk:
    outfile = VTKFile(f'output_nx{nx}_cg{cg_order}.pvf')
    outfile.write(u_1, u_2, u_exact)

# ===== Matplotlib visualization =====
if visualize:
    plot_results(u_1, u_2, u_exact, nx, ny, cg_order)

# Return results dictionary
return {
    'u_1': u_1,
    'u_2': u_2,
    'u_exact': u_exact,
    'L2_error_method1': L2_1,
    'L2_error_method2': L2_2,
    'L2_difference': L2_diff,
    'mesh_resolution': (nx, ny),
    'cg_order': cg_order
}

```

```

In [2]: def plot_results(u_1, u_2, u_exact, nx, ny, cg_order):
    """
    Create contour plots of solutions and errors.
    """

    # Create a uniform grid for plotting (works for any CG order)
    n_points = 200 # Resolution for plotting
    x_plot = np.linspace(0, 1, n_points)
    y_plot = np.linspace(0, 1, n_points)
    X, Y = np.meshgrid(x_plot, y_plot)

    # Evaluate functions at plot points
    points = np.column_stack([X.ravel(), Y.ravel()])

    try:
        u1_vals = np.array([u_1.at(p) for p in points]).reshape(X.shape)
        u2_vals = np.array([u_2.at(p) for p in points]).reshape(X.shape)
    
```

```

exact_vals = np.array([u_exact.at(p) for p in points]).reshape(X.shape)
except:
    # Fallback: use VTK interpolation (Sometimes dimension issue with higher
    from firedrake import Function, FunctionSpace
    V_plot = FunctionSpace(u_1.function_space().mesh(), 'CG', 1)
    u1_plot = Function(V_plot).interpolate(u_1)
    u2_plot = Function(V_plot).interpolate(u_2)
    exact_plot = Function(V_plot).interpolate(u_exact)

    u1_vals = np.array([u1_plot.at(p) for p in points]).reshape(X.shape)
    u2_vals = np.array([u2_plot.at(p) for p in points]).reshape(X.shape)
    exact_vals = np.array([exact_plot.at(p) for p in points]).reshape(X.shape)

# Compute errors
error_1 = u1_vals - exact_vals
error_2 = u2_vals - exact_vals

# Create figure with subplots
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
fig.suptitle(f'Poisson Equation Solution (Mesh: {nx}x{ny}, CG order: {cg_order},\n            font-size=14, font-weight='bold')

# Plot settings
levels = 20

# Row 1: Solutions
cs1 = axes[0, 0].contourf(X, Y, u1_vals, levels=levels, cmap='viridis')
axes[0, 0].set_title('Method 1: Manual Weak Form')
axes[0, 0].set_xlabel('x')
axes[0, 0].set_ylabel('y')
axes[0, 0].set_aspect('equal')
plt.colorbar(cs1, ax=axes[0, 0])

cs2 = axes[0, 1].contourf(X, Y, u2_vals, levels=levels, cmap='viridis')
axes[0, 1].set_title('Method 2: Variational Principle')
axes[0, 1].set_xlabel('x')
axes[0, 1].set_ylabel('y')
axes[0, 1].set_aspect('equal')
plt.colorbar(cs2, ax=axes[0, 1])

cs3 = axes[0, 2].contourf(X, Y, exact_vals, levels=levels, cmap='viridis')
axes[0, 2].set_title('Exact Solution')
axes[0, 2].set_xlabel('x')
axes[0, 2].set_ylabel('y')
axes[0, 2].set_aspect('equal')
plt.colorbar(cs3, ax=axes[0, 2])

# Row 2: Errors
cs4 = axes[1, 0].contourf(X, Y, error_1, levels=levels, cmap='RdBu_r')
axes[1, 0].set_title('Error: Method 1 - Exact')
axes[1, 0].set_xlabel('x')
axes[1, 0].set_ylabel('y')
axes[1, 0].set_aspect('equal')
plt.colorbar(cs4, ax=axes[1, 0])

cs5 = axes[1, 1].contourf(X, Y, error_2, levels=levels, cmap='RdBu_r')
axes[1, 1].set_title('Error: Method 2 - Exact')
axes[1, 1].set_xlabel('x')
axes[1, 1].set_ylabel('y')
axes[1, 1].set_aspect('equal')

```

```

plt.colorbar(cs5, ax=axes[1, 1])

# Difference between methods
diff = u2_vals - u1_vals
cs6 = axes[1, 2].contourf(X, Y, diff, levels=levels, cmap='RdBu_r')
axes[1, 2].set_title('Difference: Method 2 - Method 1')
axes[1, 2].set_xlabel('x')
axes[1, 2].set_ylabel('y')
axes[1, 2].set_aspect('equal')
plt.colorbar(cs6, ax=axes[1, 2])

plt.tight_layout()
plt.savefig(f'poisson_solution_nx{nx}_cg{cg_order}.png', dpi=150, bbox_inches='tight')
plt.show()

```

In [3]:

```

def convergence_study(mesh_sizes=[16, 32, 64, 128], cg_order=1):
    """
    Perform a convergence study over multiple mesh resolutions.

    Parameters:
    -----------
    mesh_sizes : List of mesh resolutions to test
    cg_order   : Order of CG elements
    """

    errors_m1 = []
    errors_m2 = []
    h_values = []

    print("\nPerforming convergence study...")

    for nx in mesh_sizes:
        result = solve_poisson(nx=nx, cg_order=cg_order, visualize=False, save_v
        errors_m1.append(result['L2_error_method1'])
        errors_m2.append(result['L2_error_method2'])
        h_values.append(1.0 / nx)

    # Plot convergence
    fig, ax = plt.subplots(figsize=(10, 6))
    ax.loglog(h_values, errors_m1, 'o-', label='Method 1', linewidth=2, markersize=10)
    ax.loglog(h_values, errors_m2, 's-', label='Method 2', linewidth=2, markersize=10)

    # Reference lines
    ax.loglog(h_values, [h**(cg_order+1) * errors_m1[0] / h_values[0]**(cg_order+1)
                         for h in h_values], '--', label=f'O(h^{cg_order+1})', alpha=0.5)

    ax.set_xlabel('Mesh size h', fontsize=12)
    ax.set_ylabel('L2 Error', fontsize=12)
    ax.set_title(f'Convergence Study (CG order {cg_order})', fontsize=14, fontweight='bold')
    ax.legend(fontsize=11)
    ax.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.savefig(f'convergence_study_cg{cg_order}.png', dpi=150, bbox_inches='tight')
    plt.show()

    return h_values, errors_m1, errors_m2

```

In [4]:

```

if __name__ == '__main__':
    # Choose what to run by setting these flags:
    RUN_SINGLE = True           # Single solve with visualization

```

```

RUN_CONVERGENCE = True      # Convergence study
RUN(CG_COMPARISON = True    # Compare CG orders

if RUN_SINGLE:
    print("\n" + "="*60)
    print("SINGLE SOLVE WITH VISUALIZATION")
    print("=*60)
    result = solve_poisson(nx=64, cg_order=1, visualize=True, save_vtk=False)

if RUN_CONVERGENCE:
    print("\n" + "="*60)
    print("CONVERGENCE STUDY")
    print("=*60)
    convergence_study(mesh_sizes=[16, 32, 64, 128], cg_order=1)

if RUN(CG_COMPARISON:
    print("\n" + "="*60)
    print("COMPARING DIFFERENT CG ORDERS")
    print("=*60)
    for order in [2, 3]:
        print(f"\n--- Testing CG order {order} ---")
        solve_poisson(nx=32, cg_order=order, visualize=True, save_vtk=False)
        convergence_study(mesh_sizes=[16, 32, 64, 128], cg_order=order)

```

SINGLE SOLVE WITH VIRTUALIZATION

Mesh resolution: 64x64, $\Delta x = 0.015625$, CG order = 1

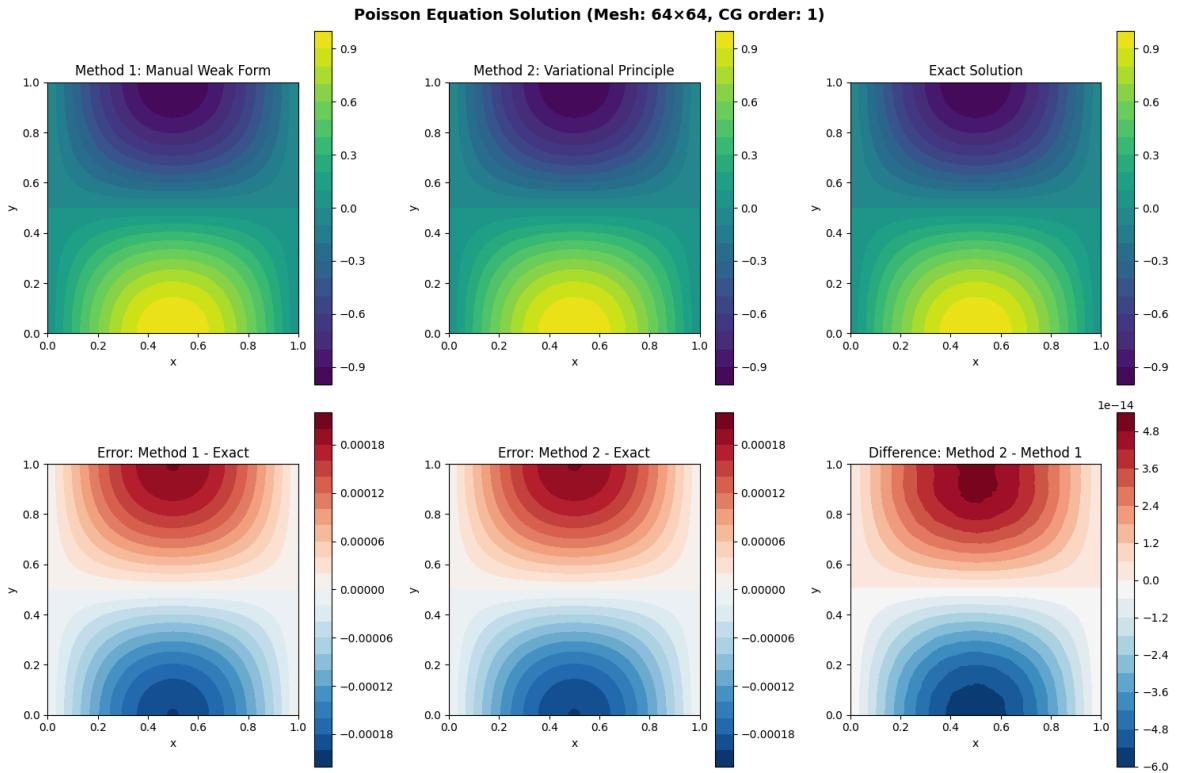
```
L2 error (Method 1 vs exact): 1.003464e-04  
L2 error (Method 2 vs exact): 1.003464e-04  
L2 norm (Method 2 - Method 1): 2.785399e-14
```

/opt/firedrake/firedrake/function.py:556: FutureWarning: The ``Function.at`` method is deprecated and will be removed in a future release. Please use the ``PointEvaluator`` class instead.

```
warnings.warn(
```

/opt/firedrake/firedrake/function.py:556: FutureWarning: The ``Function.at`` method is deprecated and will be removed in a future release. Please use the ``PointEvaluator`` class instead.

warnings.warn()



```
=====CONVERGENCE STUDY=====
```

Performing convergence study...

```
=====Mesh resolution: 16x16, Δx = 0.062500, CG order = 1=====
```

L2 error (Method 1 vs exact): 1.593011e-03

L2 error (Method 2 vs exact): 1.593011e-03

L2 norm (Method 2 - Method 1): 1.363591e-15

```
=====
```

```
=====Mesh resolution: 32x32, Δx = 0.031250, CG order = 1=====
```

L2 error (Method 1 vs exact): 4.007573e-04

L2 error (Method 2 vs exact): 4.007573e-04

L2 norm (Method 2 - Method 1): 6.159062e-15

```
=====
```

```
=====Mesh resolution: 64x64, Δx = 0.015625, CG order = 1=====
```

L2 error (Method 1 vs exact): 1.003464e-04

L2 error (Method 2 vs exact): 1.003464e-04

L2 norm (Method 2 - Method 1): 2.785399e-14

```
=====
```

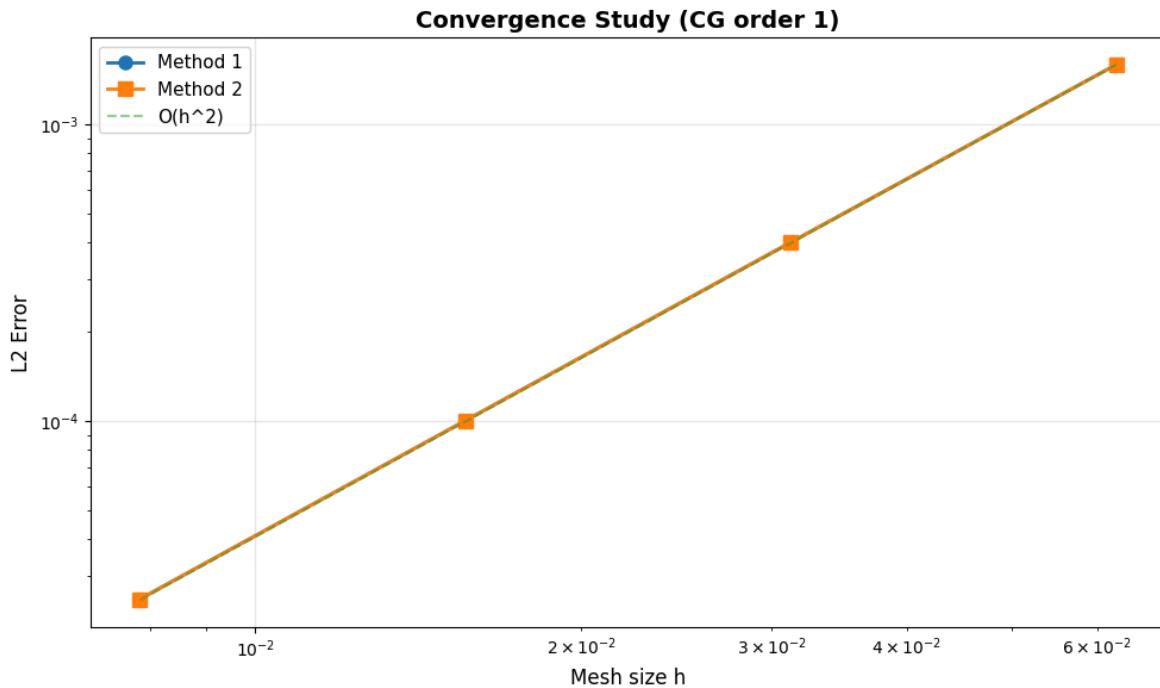
```
=====Mesh resolution: 128x128, Δx = 0.007812, CG order = 1=====
```

L2 error (Method 1 vs exact): 2.509643e-05

L2 error (Method 2 vs exact): 2.509643e-05

L2 norm (Method 2 - Method 1): 1.069251e-13

```
=====
```



```
=====
COMPARING DIFFERENT CG ORDERS
=====
```

```
--- Testing CG order 2 ---
```

```
=====
Mesh resolution: 32x32, Δx = 0.031250, CG order = 2
=====
```

```
L2 error (Method 1 vs exact): 6.527732e-08
```

```
L2 error (Method 2 vs exact): 6.527729e-08
```

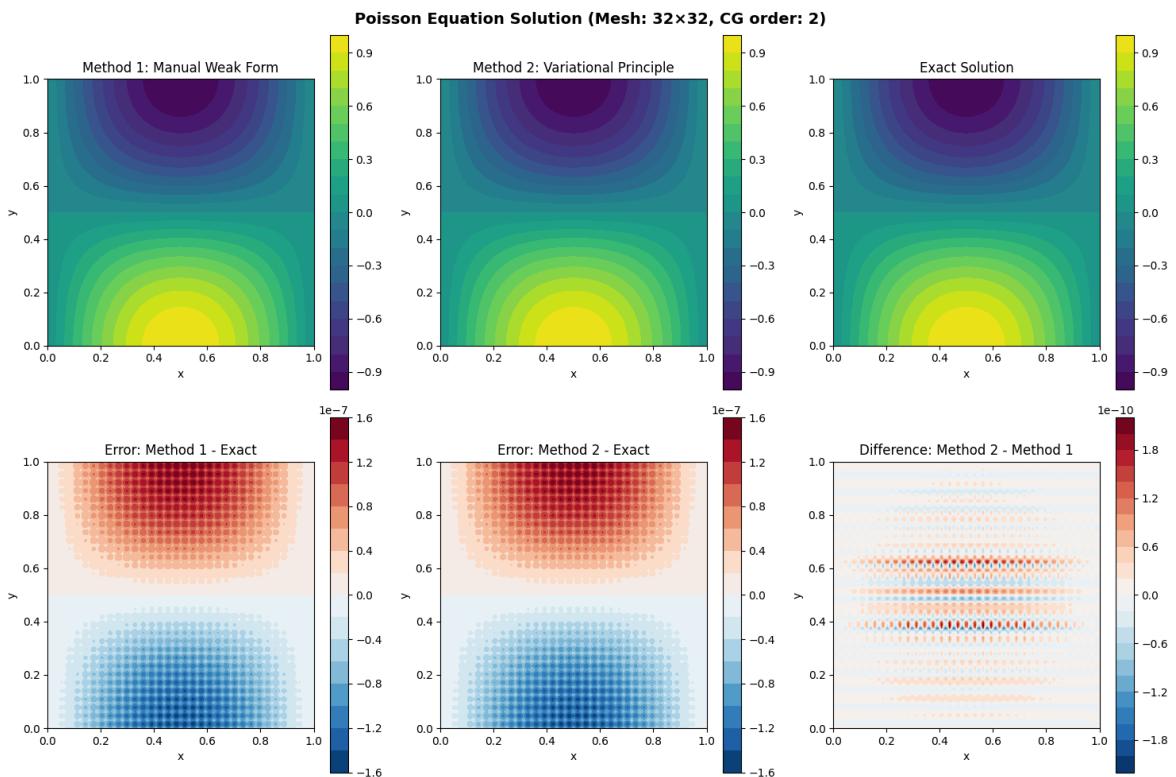
```
L2 norm (Method 2 - Method 1): 2.848572e-11
```

```
/opt/firedrake/firedrake/function.py:556: FutureWarning: The ``Function.at`` method is deprecated and will be removed in a future release. Please use the ``PointEvaluato`` class instead.
```

```
    warnings.warn(
```

```
/opt/firedrake/firedrake/function.py:556: FutureWarning: The ``Function.at`` method is deprecated and will be removed in a future release. Please use the ``PointEvaluato`` class instead.
```

```
    warnings.warn(
```



Performing convergence study...

```
=====
Mesh resolution: 16x16, Δx = 0.062500, CG order = 2
=====
```

```
L2 error (Method 1 vs exact): 1.042725e-06
=====
```

```
L2 error (Method 2 vs exact): 1.042725e-06
=====
```

```
L2 norm (Method 2 - Method 1): 6.279138e-11
=====
```

```
=====
Mesh resolution: 32x32, Δx = 0.031250, CG order = 2
=====
```

```
L2 error (Method 1 vs exact): 6.527732e-08
=====
```

```
L2 error (Method 2 vs exact): 6.527729e-08
=====
```

```
L2 norm (Method 2 - Method 1): 2.848572e-11
=====
```

```
=====
Mesh resolution: 64x64, Δx = 0.015625, CG order = 2
=====
```

```
L2 error (Method 1 vs exact): 4.081589e-09
=====
```

```
L2 error (Method 2 vs exact): 4.081519e-09
=====
```

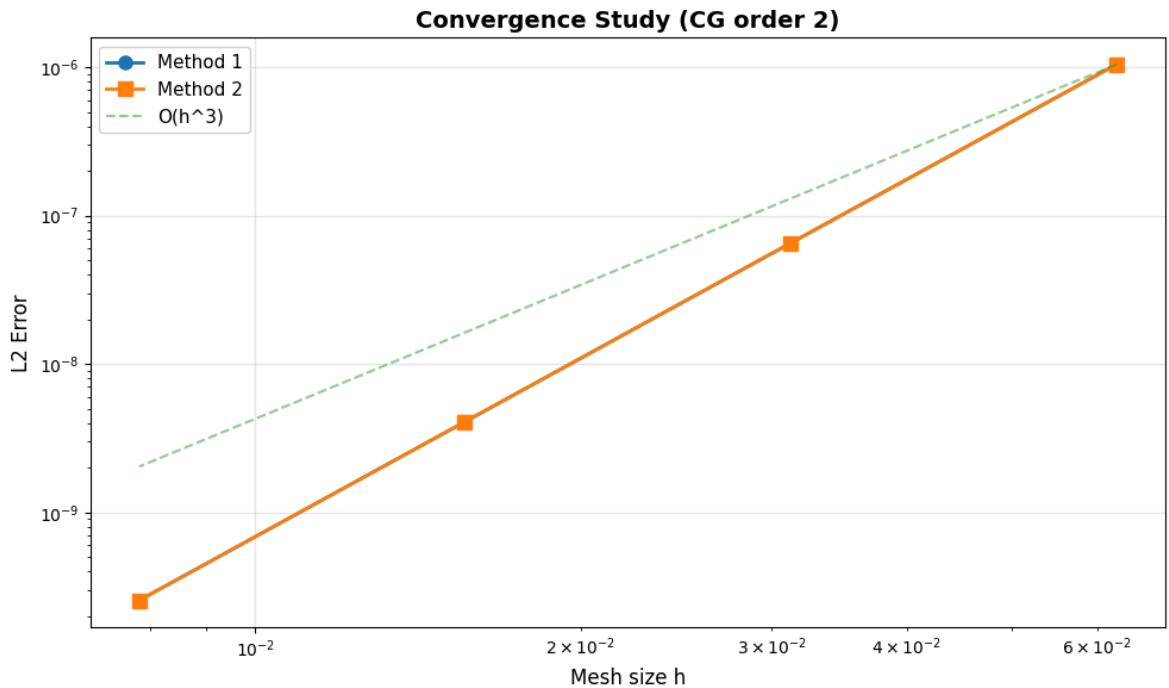
```
L2 norm (Method 2 - Method 1): 3.039484e-12
=====
```

```
=====
Mesh resolution: 128x128, Δx = 0.007812, CG order = 2
=====
```

```
L2 error (Method 1 vs exact): 2.554720e-10
=====
```

```
L2 error (Method 2 vs exact): 2.551360e-10
=====
```

```
L2 norm (Method 2 - Method 1): 2.377867e-12
=====
```



--- Testing CG order 3 ---

```
=====
Mesh resolution: 32x32, Δx = 0.031250, CG order = 3
=====
```

```
L2 error (Method 1 vs exact): 1.064388e-10
=====
```

```
L2 error (Method 2 vs exact): 1.058576e-10
=====
```

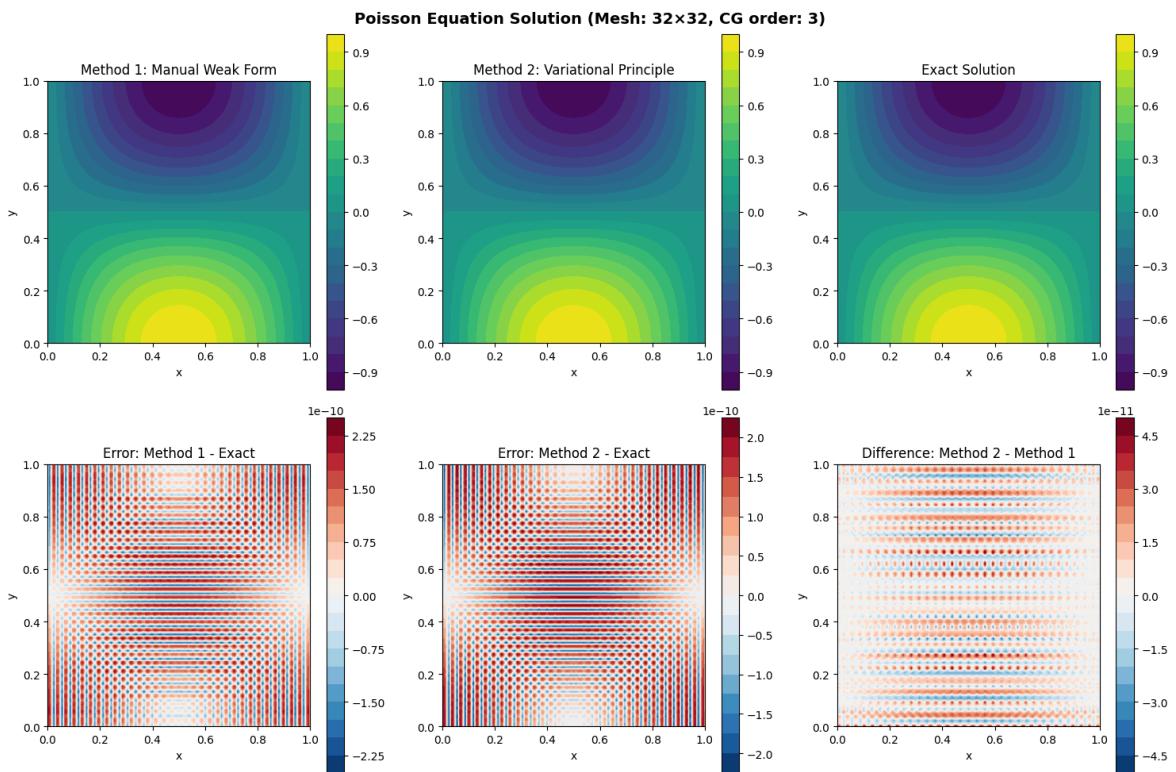
```
L2 norm (Method 2 - Method 1): 1.148145e-11
=====
```

```
/opt/firedrake/firedrake/function.py:556: FutureWarning: The ``Function.at`` method is deprecated and will be removed in a future release. Please use the ``PointEvaluato```r` class instead.
```

```
    warnings.warn(
```

```
/opt/firedrake/firedrake/function.py:556: FutureWarning: The ``Function.at`` method is deprecated and will be removed in a future release. Please use the ``PointEvaluato```r` class instead.
```

```
    warnings.warn(
```



Performing convergence study...

```
=====
Mesh resolution: 16x16, Δx = 0.062500, CG order = 3
=====
```

```
L2 error (Method 1 vs exact): 3.382140e-09
=====
```

```
L2 error (Method 2 vs exact): 3.378766e-09
=====
```

```
L2 norm (Method 2 - Method 1): 3.051000e-11
=====
```

```
=====
Mesh resolution: 32x32, Δx = 0.031250, CG order = 3
=====
```

```
L2 error (Method 1 vs exact): 1.064388e-10
=====
```

```
L2 error (Method 2 vs exact): 1.058576e-10
=====
```

```
L2 norm (Method 2 - Method 1): 1.148145e-11
=====
```

```
=====
Mesh resolution: 64x64, Δx = 0.015625, CG order = 3
=====
```

```
L2 error (Method 1 vs exact): 5.348765e-12
=====
```

```
L2 error (Method 2 vs exact): 3.340281e-12
=====
```

```
L2 norm (Method 2 - Method 1): 4.244880e-12
=====
```

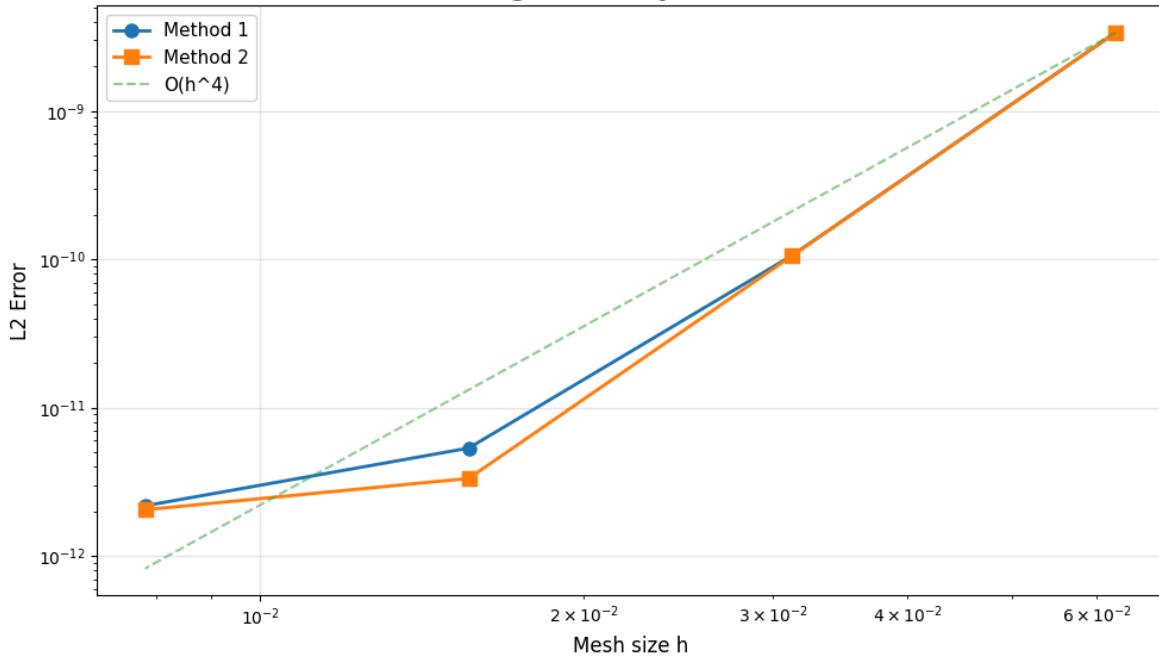
```
=====
Mesh resolution: 128x128, Δx = 0.007812, CG order = 3
=====
```

```
L2 error (Method 1 vs exact): 2.184995e-12
=====
```

```
L2 error (Method 2 vs exact): 2.053910e-12
=====
```

```
L2 norm (Method 2 - Method 1): 1.091186e-12
=====
```

Convergence Study (CG order 3)



With increasing CG order, the solution is much closer to exact at each step $x \sim 10^{-4}$ smaller with each order increased, up to CG order 3 where the machine precision is reached, further reduction in mesh size would see this error remain the same, or possibly increase due to the summation of round-off errors through the numerous computations. The convergence with mesh size is also much more significant for each order, converging with an increased order wrt mesh size with each CG order increase.

6

```
In [5]: def solve_poisson(nx=128, ny=None, cg_order=1, visualize=True, save_vtk=False):
    """
    Solve the Poisson equation using two methods and compare results.

    Parameters:
    -----
    nx      : Number of mesh cells in x-direction (default: 128)
    ny      : Number of mesh cells in y-direction (default: None, uses nx)
    cg_order : Order of continuous Galerkin elements (default: 1)
    visualize: Whether to create matplotlib plots (default: True)
    save_vtk : Whether to save VTK output for Paraview (default: True)

    Returns:
    -----
    dict : Dictionary containing solutions, errors, and mesh info
    """
    if ny is None:
        ny = nx

    # Create mesh
    mesh = UnitSquareMesh(nx, ny, quadrilateral=True)  #-----Creation of Mesh

    # Function space with specified CG order
    V = FunctionSpace(mesh, 'CG', cg_order)

    # Spatial coordinates and source term
```

```

x, y = SpatialCoordinate(mesh)

#===== New function and BCs (and exact solution) ======
f = Function(V).interpolate(2*x*(y-1)*(y-2*x+x*y+2)*exp(x-y))
u_exact = Function(V).interpolate(-x*(x-1)*y*(y-1)*exp(x-y))
bc_bottom = DirichletBC(V, Constant(0), 3)
bc_right = DirichletBC(V, Constant(0), 2)
bc_top = DirichletBC(V, Constant(0), 4)
bc_left = DirichletBC(V, Constant(0), 1)
bcs = [bc_bottom, bc_right, bc_top, bc_left]
#=====

# ===== Method 1: Weak form manually constructed - Step 1: Weak formulation
u = TrialFunction(V)
du = TestFunction(V) # Test function - equivalent to w(x,y)
a = inner(grad(u), grad(du)) * dx #
L = f * du * dx
u_1 = Function(V, name='Method_1_Manual')
solve(a == L, u_1, solver_parameters={'ksp_type': 'cg', 'pc_type': 'none'},

# ===== Method 2: Variational principle via derivative - Step 2: Ritz-Galerkin
u_2 = Function(V, name='Method_2_Variational')
Ju = (0.5*inner(grad(u_2), grad(u_2)) - u_2*f) * dx # Weak formulation - 1/2
F = derivative(Ju, u_2, du) # Build algebraic system
solve(F == 0, u_2, bcs=bcs) # Solve algebraic system

# ===== Error computation =====
L2_1 = sqrt(assemble(dot(u_1 - u_exact, u_1 - u_exact) * dx))
L2_2 = sqrt(assemble(dot(u_2 - u_exact, u_2 - u_exact) * dx))
L2_diff = sqrt(assemble(dot(u_2 - u_1, u_2 - u_1) * dx))

# Print results
print(f'\n{"*60"}')
print(f'Mesh resolution: {nx}x{ny}, Δx = {1/nx:.6f}, CG order = {cg_order}')
print(f'{{"*60"}')
print(f'L2 error (Method 1 vs exact): {L2_1:.6e}')
print(f'L2 error (Method 2 vs exact): {L2_2:.6e}')
print(f'L2 norm (Method 2 - Method 1): {L2_diff:.6e}')
print(f'{{"*60"}\n')

# ===== Save VTK for Paraview =====
if save_vtk:
    outfile = VTKFile(f'output_nx{nx}_cg{cg_order}.pvf')
    outfile.write(u_1, u_2, u_exact)

# ===== Matplotlib visualization =====
if visualize:
    plot_results(u_1, u_2, u_exact, nx, ny, cg_order)

# Return results dictionary
return {
    'u_1': u_1,
    'u_2': u_2,
    'u_exact': u_exact,
    'L2_error_method1': L2_1,
    'L2_error_method2': L2_2,
    'L2_difference': L2_diff,
    'mesh_resolution': (nx, ny),
    'cg_order': cg_order
}

```

```
In [6]: if __name__ == '__main__':
    # Choose what to run by setting these flags:
    RUN_SINGLE = True          # Single solve with visualization
    RUN_CONVERGENCE = False     # Convergence study
    RUN(CG_COMPARISON = False   # Compare CG orders

    if RUN_SINGLE:
        print("\n" + "*60)
        print("SINGLE SOLVE WITH VISUALIZATION")
        print("*60)
        result = solve_poisson(nx=64, cg_order=1, visualize=True, save_vtk=False)

    if RUN_CONVERGENCE:
        print("\n" + "*60)
        print("CONVERGENCE STUDY")
        print("*60)
        convergence_study(mesh_sizes=[16, 32, 64, 128], cg_order=1)

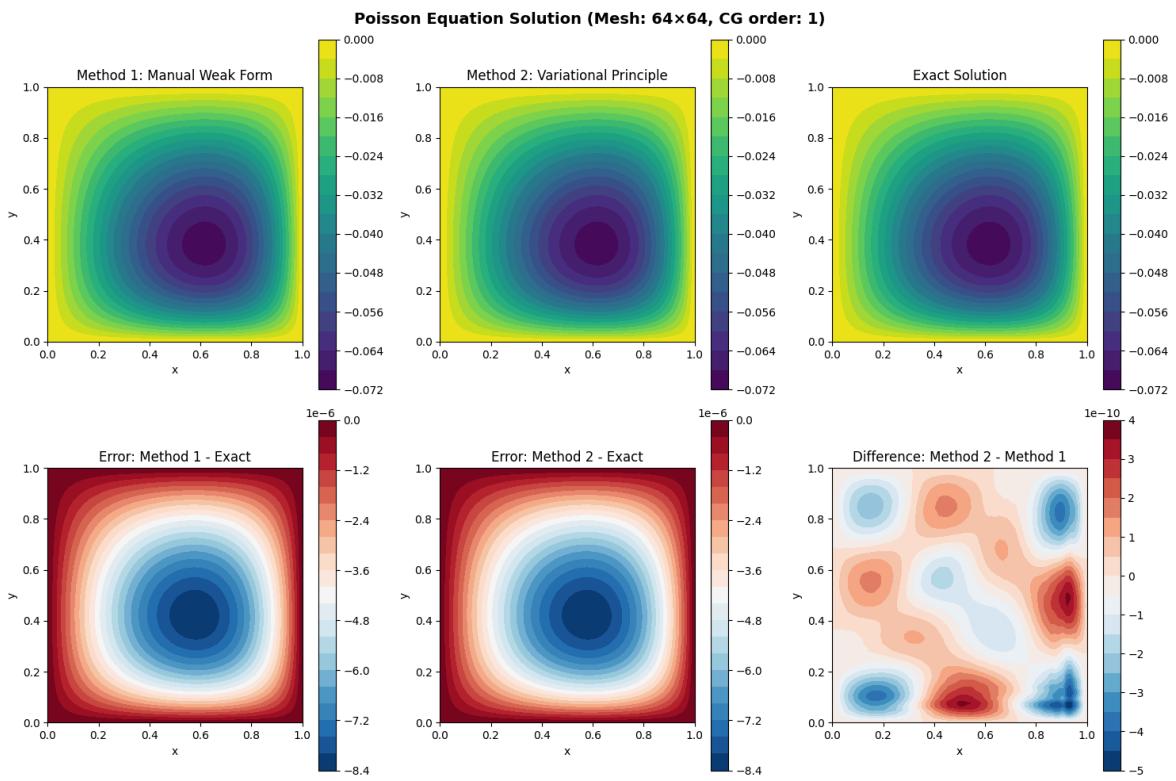
    if RUN(CG_COMPARISON:
        print("\n" + "*60)
        print("COMPARING DIFFERENT CG ORDERS")
        print("*60)
        for order in [2, 3]:
            print(f"\n--- Testing CG order {order} ---")
            solve_poisson(nx=32, cg_order=order, visualize=True, save_vtk=False)
            convergence_study(mesh_sizes=[16, 32, 64, 128], cg_order=order)
```

```
=====
SINGLE SOLVE WITH VISUALIZATION
=====

=====
Mesh resolution: 64x64, Δx = 0.015625, CG order = 1
=====

L2 error (Method 1 vs exact): 4.363992e-06
L2 error (Method 2 vs exact): 4.363993e-06
L2 norm (Method 2 - Method 1): 1.332624e-10
=====
```

```
/opt/firedrake/firedrake/function.py:556: FutureWarning: The ``Function.at`` method is deprecated and will be removed in a future release. Please use the ``PointE`` evaluator`` class instead.
    warnings.warn(
/opt/firedrake/firedrake/function.py:556: FutureWarning: The ``Function.at`` method is deprecated and will be removed in a future release. Please use the ``PointE`` evaluator`` class instead.
    warnings.warn(
```



Unsure what's causing the error to take the form it does here, whether caused by the error function or not but it is consistent through both methods (as it should be) and very small. NB: Convergence and CG order study not included here to shorten output but yields similar results to the previous example. Convergence is a little unusual for CG order 3, showing slower convergence than with the other problem. Change RUN_CONVERGENCE and RUN(CG_COMPARISON from False to True and run if you'd like to check those.

Exercise 3 - Part 2

```
In [7]: from firedrake import *
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import time as tijd

def solve_groundwater_canal(m=20, nCG=3, theta=0.5, CFL=2.3, end_time=100.0,
                           dt_output=2.0, dt_profile=10.0, rain_pattern='constant',
                           Rmax=0.000125, rain_duration=10.0, fixed_canal=False
                           ):
    """
    Solve coupled groundwater-canal system with variable rainfall.

    Parameters:
    -----
    m : Number of mesh elements
    nCG : Order of CG elements
    theta : Crank-Nicolson parameter (0.5 for CN, 1.0 for implicit, 0.0 for explicit)
    CFL : CFL number for timestep
    end_time : End time of simulation
    dt_output : Output interval for hcm and R(t)
    dt_profile : Output interval for hm profiles
    rain_pattern : 'constant', 'intermittent_1', 'intermittent_2', 'intermittent'
    """

    # Create mesh and define subdomains
    mesh = UnitSquareMesh(m, m)
    subdomains = MeshFunction("size_t", mesh, 2)
    subdomains.set_all(0)
    subdomains[0] = 1
    subdomains[1] = 2
    subdomains[2] = 3
    subdomains[3] = 4
    subdomains[4] = 5
    subdomains[5] = 6
    subdomains[6] = 7
    subdomains[7] = 8
    subdomains[8] = 9
    subdomains[9] = 10
    subdomains[10] = 11
    subdomains[11] = 12
    subdomains[12] = 13
    subdomains[13] = 14
    subdomains[14] = 15
    subdomains[15] = 16
    subdomains[16] = 17
    subdomains[17] = 18
    subdomains[18] = 19
    subdomains[19] = 20
    subdomains[20] = 21
    subdomains[21] = 22
    subdomains[22] = 23
    subdomains[23] = 24
    subdomains[24] = 25
    subdomains[25] = 26
    subdomains[26] = 27
    subdomains[27] = 28
    subdomains[28] = 29
    subdomains[29] = 30
    subdomains[30] = 31
    subdomains[31] = 32
    subdomains[32] = 33
    subdomains[33] = 34
    subdomains[34] = 35
    subdomains[35] = 36
    subdomains[36] = 37
    subdomains[37] = 38
    subdomains[38] = 39
    subdomains[39] = 40
    subdomains[40] = 41
    subdomains[41] = 42
    subdomains[42] = 43
    subdomains[43] = 44
    subdomains[44] = 45
    subdomains[45] = 46
    subdomains[46] = 47
    subdomains[47] = 48
    subdomains[48] = 49
    subdomains[49] = 50
    subdomains[50] = 51
    subdomains[51] = 52
    subdomains[52] = 53
    subdomains[53] = 54
    subdomains[54] = 55
    subdomains[55] = 56
    subdomains[56] = 57
    subdomains[57] = 58
    subdomains[58] = 59
    subdomains[59] = 60
    subdomains[60] = 61
    subdomains[61] = 62
    subdomains[62] = 63
    subdomains[63] = 64
    subdomains[64] = 65
    subdomains[65] = 66
    subdomains[66] = 67
    subdomains[67] = 68
    subdomains[68] = 69
    subdomains[69] = 70
    subdomains[70] = 71
    subdomains[71] = 72
    subdomains[72] = 73
    subdomains[73] = 74
    subdomains[74] = 75
    subdomains[75] = 76
    subdomains[76] = 77
    subdomains[77] = 78
    subdomains[78] = 79
    subdomains[79] = 80
    subdomains[80] = 81
    subdomains[81] = 82
    subdomains[82] = 83
    subdomains[83] = 84
    subdomains[84] = 85
    subdomains[85] = 86
    subdomains[86] = 87
    subdomains[87] = 88
    subdomains[88] = 89
    subdomains[89] = 90
    subdomains[90] = 91
    subdomains[91] = 92
    subdomains[92] = 93
    subdomains[93] = 94
    subdomains[94] = 95
    subdomains[95] = 96
    subdomains[96] = 97
    subdomains[97] = 98
    subdomains[98] = 99
    subdomains[99] = 100
    subdomains[100] = 101
    subdomains[101] = 102
    subdomains[102] = 103
    subdomains[103] = 104
    subdomains[104] = 105
    subdomains[105] = 106
    subdomains[106] = 107
    subdomains[107] = 108
    subdomains[108] = 109
    subdomains[109] = 110
    subdomains[110] = 111
    subdomains[111] = 112
    subdomains[112] = 113
    subdomains[113] = 114
    subdomains[114] = 115
    subdomains[115] = 116
    subdomains[116] = 117
    subdomains[117] = 118
    subdomains[118] = 119
    subdomains[119] = 120
    subdomains[120] = 121
    subdomains[121] = 122
    subdomains[122] = 123
    subdomains[123] = 124
    subdomains[124] = 125
    subdomains[125] = 126
    subdomains[126] = 127
    subdomains[127] = 128
    subdomains[128] = 129
    subdomains[129] = 130
    subdomains[130] = 131
    subdomains[131] = 132
    subdomains[132] = 133
    subdomains[133] = 134
    subdomains[134] = 135
    subdomains[135] = 136
    subdomains[136] = 137
    subdomains[137] = 138
    subdomains[138] = 139
    subdomains[139] = 140
    subdomains[140] = 141
    subdomains[141] = 142
    subdomains[142] = 143
    subdomains[143] = 144
    subdomains[144] = 145
    subdomains[145] = 146
    subdomains[146] = 147
    subdomains[147] = 148
    subdomains[148] = 149
    subdomains[149] = 150
    subdomains[150] = 151
    subdomains[151] = 152
    subdomains[152] = 153
    subdomains[153] = 154
    subdomains[154] = 155
    subdomains[155] = 156
    subdomains[156] = 157
    subdomains[157] = 158
    subdomains[158] = 159
    subdomains[159] = 160
    subdomains[160] = 161
    subdomains[161] = 162
    subdomains[162] = 163
    subdomains[163] = 164
    subdomains[164] = 165
    subdomains[165] = 166
    subdomains[166] = 167
    subdomains[167] = 168
    subdomains[168] = 169
    subdomains[169] = 170
    subdomains[170] = 171
    subdomains[171] = 172
    subdomains[172] = 173
    subdomains[173] = 174
    subdomains[174] = 175
    subdomains[175] = 176
    subdomains[176] = 177
    subdomains[177] = 178
    subdomains[178] = 179
    subdomains[179] = 180
    subdomains[180] = 181
    subdomains[181] = 182
    subdomains[182] = 183
    subdomains[183] = 184
    subdomains[184] = 185
    subdomains[185] = 186
    subdomains[186] = 187
    subdomains[187] = 188
    subdomains[188] = 189
    subdomains[189] = 190
    subdomains[190] = 191
    subdomains[191] = 192
    subdomains[192] = 193
    subdomains[193] = 194
    subdomains[194] = 195
    subdomains[195] = 196
    subdomains[196] = 197
    subdomains[197] = 198
    subdomains[198] = 199
    subdomains[199] = 200
    subdomains[200] = 201
    subdomains[201] = 202
    subdomains[202] = 203
    subdomains[203] = 204
    subdomains[204] = 205
    subdomains[205] = 206
    subdomains[206] = 207
    subdomains[207] = 208
    subdomains[208] = 209
    subdomains[209] = 210
    subdomains[210] = 211
    subdomains[211] = 212
    subdomains[212] = 213
    subdomains[213] = 214
    subdomains[214] = 215
    subdomains[215] = 216
    subdomains[216] = 217
    subdomains[217] = 218
    subdomains[218] = 219
    subdomains[219] = 220
    subdomains[220] = 221
    subdomains[221] = 222
    subdomains[222] = 223
    subdomains[223] = 224
    subdomains[224] = 225
    subdomains[225] = 226
    subdomains[226] = 227
    subdomains[227] = 228
    subdomains[228] = 229
    subdomains[229] = 230
    subdomains[230] = 231
    subdomains[231] = 232
    subdomains[232] = 233
    subdomains[233] = 234
    subdomains[234] = 235
    subdomains[235] = 236
    subdomains[236] = 237
    subdomains[237] = 238
    subdomains[238] = 239
    subdomains[239] = 240
    subdomains[240] = 241
    subdomains[241] = 242
    subdomains[242] = 243
    subdomains[243] = 244
    subdomains[244] = 245
    subdomains[245] = 246
    subdomains[246] = 247
    subdomains[247] = 248
    subdomains[248] = 249
    subdomains[249] = 250
    subdomains[250] = 251
    subdomains[251] = 252
    subdomains[252] = 253
    subdomains[253] = 254
    subdomains[254] = 255
    subdomains[255] = 256
    subdomains[256] = 257
    subdomains[257] = 258
    subdomains[258] = 259
    subdomains[259] = 260
    subdomains[260] = 261
    subdomains[261] = 262
    subdomains[262] = 263
    subdomains[263] = 264
    subdomains[264] = 265
    subdomains[265] = 266
    subdomains[266] = 267
    subdomains[267] = 268
    subdomains[268] = 269
    subdomains[269] = 270
    subdomains[270] = 271
    subdomains[271] = 272
    subdomains[272] = 273
    subdomains[273] = 274
    subdomains[274] = 275
    subdomains[275] = 276
    subdomains[276] = 277
    subdomains[277] = 278
    subdomains[278] = 279
    subdomains[279] = 280
    subdomains[280] = 281
    subdomains[281] = 282
    subdomains[282] = 283
    subdomains[283] = 284
    subdomains[284] = 285
    subdomains[285] = 286
    subdomains[286] = 287
    subdomains[287] = 288
    subdomains[288] = 289
    subdomains[289] = 290
    subdomains[290] = 291
    subdomains[291] = 292
    subdomains[292] = 293
    subdomains[293] = 294
    subdomains[294] = 295
    subdomains[295] = 296
    subdomains[296] = 297
    subdomains[297] = 298
    subdomains[298] = 299
    subdomains[299] = 300
    subdomains[300] = 301
    subdomains[301] = 302
    subdomains[302] = 303
    subdomains[303] = 304
    subdomains[304] = 305
    subdomains[305] = 306
    subdomains[306] = 307
    subdomains[307] = 308
    subdomains[308] = 309
    subdomains[309] = 310
    subdomains[310] = 311
    subdomains[311] = 312
    subdomains[312] = 313
    subdomains[313] = 314
    subdomains[314] = 315
    subdomains[315] = 316
    subdomains[316] = 317
    subdomains[317] = 318
    subdomains[318] = 319
    subdomains[319] = 320
    subdomains[320] = 321
    subdomains[321] = 322
    subdomains[322] = 323
    subdomains[323] = 324
    subdomains[324] = 325
    subdomains[325] = 326
    subdomains[326] = 327
    subdomains[327] = 328
    subdomains[328] = 329
    subdomains[329] = 330
    subdomains[330] = 331
    subdomains[331] = 332
    subdomains[332] = 333
    subdomains[333] = 334
    subdomains[334] = 335
    subdomains[335] = 336
    subdomains[336] = 337
    subdomains[337] = 338
    subdomains[338] = 339
    subdomains[339] = 340
    subdomains[340] = 341
    subdomains[341] = 342
    subdomains[342] = 343
    subdomains[343] = 344
    subdomains[344] = 345
    subdomains[345] = 346
    subdomains[346] = 347
    subdomains[347] = 348
    subdomains[348] = 349
    subdomains[349] = 350
    subdomains[350] = 351
    subdomains[351] = 352
    subdomains[352] = 353
    subdomains[353] = 354
    subdomains[354] = 355
    subdomains[355] = 356
    subdomains[356] = 357
    subdomains[357] = 358
    subdomains[358] = 359
    subdomains[359] = 360
    subdomains[360] = 361
    subdomains[361] = 362
    subdomains[362] = 363
    subdomains[363] = 364
    subdomains[364] = 365
    subdomains[365] = 366
    subdomains[366] = 367
    subdomains[367] = 368
    subdomains[368] = 369
    subdomains[369] = 370
    subdomains[370] = 371
    subdomains[371] = 372
    subdomains[372] = 373
    subdomains[373] = 374
    subdomains[374] = 375
    subdomains[375] = 376
    subdomains[376] = 377
    subdomains[377] = 378
    subdomains[378] = 379
    subdomains[379] = 380
    subdomains[380] = 381
    subdomains[381] = 382
    subdomains[382] = 383
    subdomains[383] = 384
    subdomains[384] = 385
    subdomains[385] = 386
    subdomains[386] = 387
    subdomains[387] = 388
    subdomains[388] = 389
    subdomains[389] = 390
    subdomains[390] = 391
    subdomains[391] = 392
    subdomains[392] = 393
    subdomains[393] = 394
    subdomains[394] = 395
    subdomains[395] = 396
    subdomains[396] = 397
    subdomains[397] = 398
    subdomains[398] = 399
    subdomains[399] = 400
    subdomains[400] = 401
    subdomains[401] = 402
    subdomains[402] = 403
    subdomains[403] = 404
    subdomains[404] = 405
    subdomains[405] = 406
    subdomains[406] = 407
    subdomains[407] = 408
    subdomains[408] = 409
    subdomains[409] = 410
    subdomains[410] = 411
    subdomains[411] = 412
    subdomains[412] = 413
    subdomains[413] = 414
    subdomains[414] = 415
    subdomains[415] = 416
    subdomains[416] = 417
    subdomains[417] = 418
    subdomains[418] = 419
    subdomains[419] = 420
    subdomains[420] = 421
    subdomains[421] = 422
    subdomains[422] = 423
    subdomains[423] = 424
    subdomains[424] = 425
    subdomains[425] = 426
    subdomains[426] = 427
    subdomains[427] = 428
    subdomains[428] = 429
    subdomains[429] = 430
    subdomains[430] = 431
    subdomains[431] = 432
    subdomains[432] = 433
    subdomains[433] = 434
    subdomains[434] = 435
    subdomains[435] = 436
    subdomains[436] = 437
    subdomains[437] = 438
    subdomains[438] = 439
    subdomains[439] = 440
    subdomains[440] = 441
    subdomains[441] = 442
    subdomains[442] = 443
    subdomains[443] = 444
    subdomains[444] = 445
    subdomains[445] = 446
    subdomains[446] = 447
    subdomains[447] = 448
    subdomains[448] = 449
    subdomains[449] = 450
    subdomains[450] = 451
    subdomains[451] = 452
    subdomains[452] = 453
    subdomains[453] = 454
    subdomains[454] = 455
    subdomains[455] = 456
    subdomains[456] = 457
    subdomains[457] = 458
    subdomains[458] = 459
    subdomains[459] = 460
    subdomains[460] = 461
    subdomains[461] = 462
    subdomains[462] = 463
    subdomains[463] = 464
    subdomains[464] = 465
    subdomains[465] = 466
    subdomains[466] = 467
    subdomains[467] = 468
    subdomains[468] = 469
    subdomains[469] = 470
    subdomains[470] = 471
    subdomains[471] = 472
    subdomains[472] = 473
    subdomains[473] = 474
    subdomains[474] = 475
    subdomains[475] = 476
    subdomains[476] = 477
    subdomains[477] = 478
    subdomains[478] = 479
    subdomains[479] = 480
    subdomains[480] = 481
    subdomains[481] = 482
    subdomains[482] = 483
    subdomains[483] = 484
    subdomains[484] = 485
    subdomains[485] = 486
    subdomains[486] = 487
    subdomains[487] = 488
    subdomains[488] = 489
    subdomains[489] = 490
    subdomains[490] = 491
    subdomains[491] = 492
    subdomains[492] = 493
    subdomains[493] = 494
    subdomains[494] = 495
    subdomains[495] = 496
    subdomains[496] = 497
    subdomains[497] = 498
    subdomains[498] = 499
    subdomains[499] = 500
    subdomains[500] = 501
    subdomains[501] = 502
    subdomains[502] = 503
    subdomains[503] = 504
    subdomains[504] = 505
    subdomains[505] = 506
    subdomains[506] = 507
    subdomains[507] = 508
    subdomains[508] = 509
    subdomains[509] = 510
    subdomains[510] = 511
    subdomains[511] = 512
    subdomains[512] = 513
    subdomains[513] = 514
    subdomains[514] = 515
    subdomains[515] = 516
    subdomains[516] = 517
    subdomains[517] = 518
    subdomains[518] = 519
    subdomains[519] = 520
    subdomains[520] = 521
    subdomains[521] = 522
    subdomains[522] = 523
    subdomains[523] = 524
    subdomains[524] = 525
    subdomains[525] = 526
    subdomains[526] = 527
    subdomains[527] = 528
    subdomains[528] = 529
    subdomains[529] = 530
    subdomains[530] = 531
    subdomains[531] = 532
    subdomains[532] = 533
    subdomains[533] = 534
    subdomains[534] = 535
    subdomains[535] = 536
    subdomains[536] = 537
    subdomains[537] = 538
    subdomains[538] = 539
    subdomains[539] = 540
    subdomains[540] = 541
    subdomains[541] = 542
    subdomains[542] = 543
    subdomains[543] = 544
    subdomains[544] = 545
    subdomains[545] = 546
    subdomains[546] = 547
    subdomains[547] = 548
    subdomains[548] = 549
    subdomains[549] = 550
    subdomains[550] = 551
    subdomains[551] = 552
    subdomains[552] = 553
    subdomains[553] = 554
    subdomains[554] = 555
    subdomains[555] = 556
    subdomains[556] = 557
    subdomains[557] = 558
    subdomains[558] = 559
    subdomains[559] = 560
    subdomains[560] = 561
    subdomains[561] = 562
    subdomains[562] = 563
    subdomains[563] = 564
    subdomains[564] = 565
    subdomains[565] = 566
    subdomains[566] = 567
    subdomains[567] = 568
    subdomains[568] = 569
    subdomains[569] = 570
    subdomains[570] = 571
    subdomains[571] = 572
    subdomains[572] = 573
    subdomains[573] = 574
    subdomains[574] = 575
    subdomains[575] = 576
    subdomains[576] = 577
    subdomains[577] = 578
    subdomains[578] = 579
    subdomains[579] = 580
    subdomains[580] = 581
    subdomains[581] = 582
    subdomains[582] = 583
    subdomains[583] = 584
    subdomains[584] = 585
    subdomains[585] = 586
    subdomains[586] = 587
    subdomains[587] = 588
    subdomains[588] = 589
    subdomains[589] = 590
    subdomains[590] = 591
    subdomains[591] = 592
    subdomains[592] = 593
    subdomains[593] = 594
    subdomains[594] = 595
    subdomains[595] = 596
    subdomains[596] = 597
    subdomains[597] = 598
    subdomains[598] = 599
    subdomains[599] = 600
    subdomains[600] = 601
    subdomains[601] = 602
    subdomains[602] = 603
    subdomains[603] = 604
    subdomains[604] = 605
    subdomains[605] = 606
    subdomains[606] = 607
    subdomains[607] = 608
    subdomains[608] = 609
    subdomains[609] = 610
    subdomains[610] = 611
    subdomains[611] = 612
    subdomains[612] = 613
    subdomains[613] = 614
    subdomains[614] = 615
    subdomains[615] = 616
    subdomains[616] = 617
    subdomains[617] = 618
    subdomains[618] = 619
    subdomains[619] = 620
    subdomains[620] = 621
    subdomains[621] = 622
    subdomains[622] = 623
    subdomains[623] = 624
    subdomains[624] = 625
    subdomains[625] = 626
    subdomains[626] = 627
    subdomains[627] = 628
    subdomains[628] = 629
    subdomains[629] = 630
    subdomains[630] = 631
    subdomains[631] = 632
    subdomains[632] = 633
    subdomains[633] = 634
    subdomains[634] = 635
    subdomains[635] = 636
    subdomains[636] = 637
    subdomains[637] = 638
    subdomains[638] = 639
    subdomains[639] = 640
    subdomains[640] = 641
    subdomains[641] = 642
    subdomains[642] = 643
    subdomains[643] = 644
    subdomains[644] = 645
    subdomains[645] = 646
    subdomains[646] = 647
    subdomains[647] = 648
    subdomains[648] = 649
    subdomains[649] = 650
    subdomains[650] = 651
    subdomains[651] = 652
    subdomains[652] = 653
    subdomains[653] = 654
    subdomains[654] = 655
    subdomains[655] = 656
    subdomains[656] = 657
    subdomains[657] = 658
    subdomains[658] = 659
    subdomains[659] = 660
    subdomains[660] = 661
    subdomains[661] = 662
    subdomains[662] = 663
    subdomains[663] = 664
    subdomains[664] = 665
    subdomains[665] = 666
    subdomains[666] = 667
    subdomains[667] = 668
    subdomains[668] = 669
    subdomains[669] = 670
    subdomains[670] = 671
    subdomains[671] = 672
    subdomains[672] = 673
    subdomains[673] = 674
    subdomains[674] = 675
    subdomains[675] = 676
    subdomains[676] = 677
    subdomains[677] = 678
    subdomains[678] = 679
    subdomains[679] = 680
    subdomains[680] = 681
    subdomains[681] = 682
    subdomains[682] = 683
    subdomains[683] = 684
    subdomains[684] = 685
    subdomains[685] = 686
    subdomains[686] = 687
    subdomains[687] = 688
    subdomains[688] = 689
    subdomains[689] = 690
    subdomains[690] = 691
    subdomains[691] = 692
    subdomains[692] = 693
    subdomains[693] = 694
    subdomains[694] = 695
    subdomains[695] = 696
    subdomains[696] = 697
    subdomains[697] = 698
    subdomains[698] = 699
    subdomains[699] = 700
    subdomains[700] = 701
    subdomains[701] = 702
    subdomains[702] = 703
    subdomains[703] = 704
    subdomains[704] = 705
    subdomains[705] = 706
    subdomains[706] = 707
    subdomains[707] = 708
    subdomains[708] = 709
    subdomains[709] = 710
    subdomains[710] = 711
    subdomains[711] = 712
    subdomains[712] = 713
    subdomains[713] = 714
    subdomains[714] = 715
    subdomains[715] = 716
    subdomains[716] = 717
    subdomains[717] = 718
    subdomains[718] = 719
    subdomains[719] = 720
    subdomains[720] = 721
    subdomains[721] = 722
    subdomains[722] = 723
    subdomains[723] = 724
    subdomains[724] = 725
    subdomains[725] = 726
    subdomains[726] = 727
    subdomains[727] = 728
    subdomains[728] = 729
    subdomains[729] = 730
    subdomains[730] = 731
    subdomains[731] = 732
    subdomains[732] = 733
    subdomains[733] = 734
    subdomains[734] = 735
    subdomains[735] = 736
    subdomains[736] = 737
    subdomains[737] = 738
    subdomains[738] = 739
    subdomains[739] = 740
    subdomains[740] = 741
    subdomains[741] = 742
    subdomains[742] = 743
    subdomains[743] = 744
    subdomains[744] = 745
    subdomains[745] = 746
    subdomains[746] = 747
    subdomains[747] = 748
    subdomains[748
```

```

Rmax : Maximum rain rate
rain_duration : Duration of rain cycle (default 10s)
fixed_canal : If True, fix hcm = 0.07m (simpler problem)

Returns:
-----
dict : Solution data and mesh information
"""

# Mesh setup
Ly = 0.85
dy = Ly/m
mesh = IntervalMesh(m, 0, Ly)
y, = SpatialCoordinate(mesh)

# Function space
V = FunctionSpace(mesh, "CG", nCG)

# Timestep
Dt = CFL * 0.5 * dy * dy
dt = Constant(Dt)

# Physical parameters
mpor = 0.3
sigma = 0.8
Lc = 0.05
kperm = 1e-8
nu = 1.0e-6
g = 9.81
alpha = kperm / (nu * mpor * sigma)
gam = Lc / (mpor * sigma)
fac2 = sqrt(g) / (mpor * sigma)

# Rain function
def get_rain_rate(t_current):
    """Get rain rate at current time based on pattern"""
    if rain_pattern == 'constant':
        return Rmax
    elif rain_pattern.startswith('intermittent'):
        # Extract duration from pattern name
        duration_map = {
            'intermittent_1': 1.0,
            'intermittent_2': 2.0,
            'intermittent_4': 4.0,
            'intermittent_9': 9.0
        }
        rain_on_duration = duration_map.get(rain_pattern, 0.0)
        cycle_time = t_current % rain_duration
        return Rmax if cycle_time < rain_on_duration else 0.0
    else:
        return Rmax

# Initial conditions
h_prev = Function(V, name="hm"). interpolate(0.0 + 0.0*y) # hm(y, 0) = 0
hcm_prev = 0.0 # hcm(0) = 0

# Storage for time series
times = []
hcm_values = []
R_values = []

```

```

hm_profiles = []
profile_times = []

# Create output file
outfile = VTKFile("./Results/groundwater_canal.pvd")

# Flux function
def flux(h, phi, R_current):
    return (alpha * g * h * dot(grad(h), grad(phi)) -
            (R_current * phi) / (mpor * sigma))

# Setup solver
t = 0.0
t_next_output = dt_output
t_next_profile = dt_profile
t_next_print = 10.0 # Print every 10 seconds

print(f"\n{'='*70}")
print("Groundwater-Canal System Simulation")
print(f"\n{'='*70}")
print(f"Mesh elements: {m}, CG order: {nCG}, theta: {theta}")
print(f"Timestep: {Dt:.6e} s, End time: {end_time} s")
print(f"Rain pattern: {rain_pattern}, Rmax: {Rmax}")
print(f"Fixed canal: {fixed_canal}")
print(f"\n{'='*70}\n")

# Initial output
times.append(0.0)
hcm_values.append(hcm_prev)
R_values.append(get_rain_rate(0.0))
hm_profiles.append(h_prev.dat.data.copy())
profile_times.append(0.0)
outfile.write(h_prev, time=0.0)

# Time stepping
step = 0
start_time = tijd.time()

if fixed_canal:
    # Simpler problem with fixed canal height
    hcm_fixed = 0.07
    h = Function(V)
    h.assign(h_prev)

    while t < end_time:
        t += Dt
        step += 1

        R_current = get_rain_rate(t)

        # Variational formulation
        phi = TestFunction(V)
        F = ((h - h_prev) * phi / dt +
              theta * flux(h, phi, R_current) +
              (1 - theta) * flux(h_prev, phi, R_current)) * dx

        # Dirichlet BC at y=0
        bc1 = DirichletBC(V, hcm_fixed, 1)

        # Solve

```

```

solve(F == 0, h, bcs=bc1,
      solver_parameters={'ksp_type': 'preonly', 'pc_type': 'lu'})

h_prev.assign(h)
hcm_current = hcm_fixed

# Store outputs
if t >= t_next_output:
    times.append(t)
    hcm_values.append(hcm_current)
    R_values.append(R_current)
    t_next_output += dt_output

if t >= t_next_profile:
    hm_profiles.append(h_prev.dat.data.copy())
    profile_times.append(t)
    t_next_profile += dt_profile
    print(f"t = {t:.2f}s, hcm = {hcm_current:.6f}m, R = {R_current:.6f}m")
    outfile.write(h_prev, time=t)

else:
    # Full problem with canal equation
    hcm_current = hcm_prev

    # Setup variational problem once (outside loop for efficiency)
    phi = TestFunction(V)
    R_const = Constant(Rmax) # Use Constant for efficient updating

    if theta == 0.0:
        # EXPLICIT scheme with weir boundary
        h, out = TrialFunction(V), Function(V)
        out.assign(h_prev)

        aa = (h * phi / dt) * dx + (gam * phi * h / dt) * ds(1)
        L = (h_prev * phi / dt - flux(h_prev, phi, R_const)) * dx
        L += (gam * phi * h_prev / dt - phi * fac2 * max_value(2.0*h_prev/3.0, 0.0)) * ds(1)

    # Create problem and solver once
    problem = LinearVariationalProblem(aa, L, out)
    solver = LinearVariationalSolver(problem,
                                     solver_parameters={'mat_type': 'aij',
                                                        'ksp_type': 'preonly',
                                                        'pc_type': 'lu',
                                                        'pc_factor_mat_solv': 'jacobi',
                                                        'ksp_rtol': 1e-14})

    while t < end_time:
        t += Dt
        step += 1

        R_current = get_rain_rate(t)
        R_const.assign(R_current)

        # Solve groundwater equation
        solver.solve()

        # Update canal height using canal equation
        h_at_boundary = out.dat.data[0]
        q_in = fac2 * max(2.0 * h_at_boundary / 3.0, 0.0)**1.5
        q_out = fac2 * max(2.0 * hcm_current / 3.0, 0.0)**1.5

```

```

dhcm_dt = (q_in - q_out) / Lc
hcm_current = hcm_prev + dhcm_dt * Dt
hcm_current = max(hcm_current, 0.0)

# Update for next timestep
h_prev.assign(out)
hcm_prev = hcm_current

# Store outputs
if t >= t_next_output - 1e-10:
    times.append(t)
    hcm_values.append(hcm_current)
    R_values.append(R_current)
    t_next_output += dt_output

if t >= t_next_profile - 1e-10:
    hm_profiles.append(h_prev.dat.data.copy())
    profile_times.append(t)
    outfile.write(h_prev, time=t)
    t_next_profile += dt_profile

# Print output every 10 seconds
if t >= t_next_print - 1e-10:
    print(f"t = {t:.2f}s, hcm = {hcm_current:.6f}m, R = {R_current:.6f}mm")
    t_next_print += 10.0

else:
    # IMPLICIT/CRANK-NICOLSON scheme with weir boundary - Newton iteration
    h = Function(V)
    h.assign(h_prev)

    F = ((h - h_prev) * phi / dt +
          theta * flux(h, phi, R_const) +
          (1 - theta) * flux(h_prev, phi, R_const)) * dx

    # Boundary contribution at y=0 with weir equation
    F_boundary = (gam * phi * (h - h_prev) / dt +
                  theta * phi * fac2 * max_value(2.0 * h / 3.0, 0.0)**1.5
                  (1 - theta) * phi * fac2 * max_value(2.0 * h_prev / 3.0, 0.0)**1.5

    # Create problem and solver once
    problem = NonlinearVariationalProblem(F + F_boundary, h)
    solver = NonlinearVariationalSolver(problem,
                                         solver_parameters={'mat_type': 'aij',
                                                             'ksp_type': 'preonly',
                                                             'pc_type': 'lu',
                                                             'pc_factor_mat_solver_package': 'mumps',
                                                             'ksp_rtol': 1e-1})

    while t < end_time:
        t += Dt
        step += 1

        R_current = get_rain_rate(t)
        R_const.assign(R_current)

        # Solve groundwater equation
        solver.solve()

```

```

# Update canal height using canal equation
h_at_boundary = h.dat.data[0]
q_in = fac2 * max(2.0 * h_at_boundary / 3.0, 0.0)**1.5
q_out = fac2 * max(2.0 * hcm_current / 3.0, 0.0)**1.5

dhcm_dt = (q_in - q_out) / Lc
hcm_current = hcm_prev + dhcm_dt * Dt
hcm_current = max(hcm_current, 0.0)

# Update for next timestep
h_prev.assign(h)
hcm_prev = hcm_current

# Store outputs
if t >= t_next_output - 1e-10:
    times.append(t)
    hcm_values.append(hcm_current)
    R_values.append(R_current)
    t_next_output += dt_output

if t >= t_next_profile - 1e-10:
    hm_profiles.append(h_prev.dat.data.copy())
    profile_times.append(t)
    outfile.write(h_prev, time=t)
    t_next_profile += dt_profile

# Print output every 10 seconds
if t >= t_next_print - 1e-10:
    print(f"t = {t:.2f}s, hcm = {hcm_current:.6f}m, R = {R_current:.6f}m")
    t_next_print += 10.0

elapsed_time = tijd.time() - start_time
print(f"\n{'='*70}")
print(f"Simulation completed in {elapsed_time:.2f} seconds ({step} steps)")
print(f"Final time: {t:.2f}s")
print(f"Final hcm: {hcm_values[-1]:.6f}m")
print(f"Final hm at y=L: {h_prev.dat.data[-1]:.6f}m")
print(f"{'='*70}\n")

# Get mesh coordinates
y_dofs = Function(V).interpolate(y).dat.data.copy()

return {
    'times': np.array(times),
    'hcm': np.array(hcm_values),
    'R': np.array(R_values),
    'hm_profiles': hm_profiles,
    'profile_times': profile_times,
    'y_coords': y_dofs,
    'final_hm': h_prev.dat.data.copy(),
    'mesh_size': m,
    'cg_order': nCG,
    'timestep': Dt,
    'rain_pattern': rain_pattern
}

def plot_results(result, save_prefix=''):
    """Plot time series and profiles"""

```

```

fig = plt.figure(figsize=(16, 10))

# Time series plots
ax1 = plt.subplot(2, 3, 1)
ax1.plot(result['times'], result['hcm'], 'b-', linewidth=2)
ax1.set_xlabel('Time (s)', fontsize=12)
ax1.set_ylabel('Canal height hcm (m)', fontsize=12)
ax1.set_title('Canal Height vs Time', fontsize=13, fontweight='bold')
ax1.grid(True, alpha=0.3)

ax2 = plt.subplot(2, 3, 2)
ax2.plot(result['times'], result['R'], 'r-', linewidth=2)
ax2.set_xlabel('Time (s)', fontsize=12)
ax2.set_ylabel('Rain rate R (m/s)', fontsize=12)
ax2.set_title('Rain Rate vs Time', fontsize=13, fontweight='bold')
ax2.grid(True, alpha=0.3)

# Profiles at different times
ax3 = plt.subplot(2, 3, 3)
cmap = plt.cm.viridis
colors = [cmap(i/len(result['profile_times'])) for i in range(len(result['profile_times']))]

for i, (hm, t) in enumerate(zip(result['hm_profiles'], result['profile_times'])):
    ax3.plot(result['y_coords'], hm, color=colors[i],
             label=f't={t:.0f}s', linewidth=1.5)

ax3.set_xlabel('Position y (m)', fontsize=12)
ax3.set_ylabel('Groundwater height hm (m)', fontsize=12)
ax3.set_title('Groundwater Profiles', fontsize=13, fontweight='bold')
ax3.legend(fontsize=8, ncol=2)
ax3.grid(True, alpha=0.3)

# Final profile detail
ax4 = plt.subplot(2, 3, 4)
ax4.plot(result['y_coords'], result['final_hm'], 'b-', linewidth=2, marker='o')
ax4.set_xlabel('Position y (m)', fontsize=12)
ax4.set_ylabel('Groundwater height hm (m)', fontsize=12)
ax4.set_title(f'Final Profile at t={result["times"][-1]:.0f}s', fontsize=13,
             fontweight='bold')
ax4.grid(True, alpha=0.3)

# Combined plot: hcm and R
ax5 = plt.subplot(2, 3, 5)
ax5_twin = ax5.twinx()

l1 = ax5.plot(result['times'], result['hcm'], 'b-', linewidth=2, label='hcm')
l2 = ax5_twin.plot(result['times'], result['R'], 'r-', linewidth=2, label='R')

ax5.set_xlabel('Time (s)', fontsize=12)
ax5.set_ylabel('Canal height hcm (m)', fontsize=12, color='b')
ax5_twin.set_ylabel('Rain rate R (m/s)', fontsize=12, color='r')
ax5.tick_params(axis='y', labelcolor='b')
ax5_twin.tick_params(axis='y', labelcolor='r')
ax5.set_title('Canal & Rain vs Time', fontsize=13, fontweight='bold')
ax5.grid(True, alpha=0.3)

# Info text
ax6 = plt.subplot(2, 3, 6)
ax6.axis('off')

info_text = """

```

Simulation Parameters:

```

Mesh elements: {result['mesh_size']}
CG order: {result['cg_order']}
Timestep: {result['timestep']} s
Rain pattern: {result['rain_pattern']}

Final Results (t={result['times'][-1]:.1f}s):
Final hcm: {result['hcm'][-1]:.6f} m
Final hm(0): {result['final_hm'][0]:.6f} m
Final hm(Ly): {result['final_hm'][-1]:.6f} m

Steady State Check:
Δhcm (last 20s): {abs(result['hcm'][-1] - result['hcm'][-10]):.6e} m
"""

ax6.text(0.1, 0.5, info_text, fontsize=11, family='monospace',
         verticalalignment='center', bbox=dict(boxstyle='round',
         facecolor='wheat', alpha=0.3))

plt.suptitle(f'Groundwater-Canal System: {result["rain_pattern"]}',
             fontsize=15, fontweight='bold')
plt.tight_layout()

if save_prefix:
    plt.savefig(f'{save_prefix}_{result["rain_pattern"]}.png', dpi=150, bbox
    plt.show()

```

In [24]:

```

def convergence_study(mesh_sizes, fixed_canal=True, theta=0.5):
    """Study mesh convergence"""

    print("\n" + "="*70)
    print("CONVERGENCE STUDY")
    print("=*70 + "\n")

    solutions = []
    errors = []
    h_values = []

    # Reference solution (finest mesh)
    print("Computing reference solution (m=160)...")
    ref_result = solve_groundwater_canal(m=160, nCG=3, theta=theta, end_time=10.
                                          dt_output=10.0, dt_profile=10.0,
                                          fixed_canal=fixed_canal)

    print("\nComputing solutions for convergence study:")
    for m in mesh_sizes:
        print(f"\nSolving with m={m}...")
        result = solve_groundwater_canal(m=m, nCG=3, theta=theta, end_time=10.0,
                                         dt_output=10.0, dt_profile=10.0,
                                         fixed_canal=fixed_canal)

        # Interpolate to common grid for comparison
        y_ref = ref_result['y_coords']
        y_test = result['y_coords']
        hm_ref = ref_result['final_hm']
        hm_test = np.interp(y_ref, y_test, result['final_hm'])

```

```

# Compute L2 error
error = np.sqrt(np.trapz((hm_test - hm_ref)**2, y_ref))
errors.append(error)
h_values.append(1.0/m)
solutions.append(result)

print(f" Mesh {m}: L2 error = {error:.6e}, final hcm = {result['hcm'][0]}")

# Plot convergence
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Convergence plot
ax1.loglog(h_values, errors, 'o-', linewidth=2, markersize=8, label='Compute')

# Reference Lines for different orders
if len(errors) >= 2:
    # Estimate convergence rate
    rate = np.log(errors[-1]/errors[0]) / np.log(h_values[-1]/h_values[0])
    print(f"\nEstimated convergence rate: {rate:.2f}")

    # Plot reference line with estimated rate
    ax1.loglog(h_values, [errors[0] * (h/h_values[0])**2 for h in h_values],
               '--', label='O(h^2)', alpha=0.5, color='red')
    ax1.loglog(h_values, [errors[0] * (h/h_values[0])**rate for h in h_values],
               ':', label=f'O(h^{rate:.2f})', alpha=0.7, color='green')

ax1.set_xlabel('Mesh size h', fontsize=12)
ax1.set_ylabel('L2 Error', fontsize=12)
ax1.set_title(f'Convergence Study at t=100s (θ={theta})', fontsize=13, fontweight='bold')
ax1.legend(fontsize=11)
ax1.grid(True, alpha=0.3)

# Compare final profiles
for result in solutions:
    ax2.plot(result['y_coords'], result['final_hm'],
              label=f'm={result["mesh_size"]}', linewidth=2)

# Add reference solution
ax2.plot(ref_result['y_coords'], ref_result['final_hm'],
          'k--', label='Reference (m=160)', linewidth=2, alpha=0.5)

ax2.set_xlabel('Position y (m)', fontsize=12)
ax2.set_ylabel('hm (m)', fontsize=12)
ax2.set_title('Final Profiles (t=100s)', fontsize=13, fontweight='bold')
ax2.legend(fontsize=11)
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig(f'convergence_study_theta{theta}.png', dpi=150, bbox_inches='tight')
plt.show()

print("\n" + "="*70)
print("CONVERGENCE STUDY COMPLETE")
print("="*70)

return solutions, errors, h_values

```

In [9]: `def compare_rain_patterns():
 """Compare different rain patterns"""`

```

print("\n" + "="*70)
print("COMPARING RAIN PATTERNS")
print("="*70 + "\n")

patterns = ['constant', 'intermittent_1', 'intermittent_2',
            'intermittent_4', 'intermittent_9']

results = {}

for pattern in patterns:
    print(f"\nSolving with pattern: {pattern}")
    result = solve_groundwater_canal(m=40, nCG=3, end_time=100.0,
                                      dt_output=2.0, dt_profile=10.0,
                                      rain_pattern=pattern, fixed_canal=False)
    results[pattern] = result

# Comparative plots
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Canal height comparison
ax1 = axes[0, 0]
for pattern, result in results.items():
    label = pattern.replace('intermittent_', '').replace('constant', 'consta'
    if pattern != 'constant':
        label = label + '/10s'
    ax1.plot(result['times'], result['hcm'], linewidth=2, label=label)

    ax1.set_xlabel('Time (s)', fontsize=12)
    ax1.set_ylabel('Canal height hcm (m)', fontsize=12)
    ax1.set_title('Canal Height: Different Rain Patterns', fontsize=13, fontweight='bold')
    ax1.legend(fontsize=10)
    ax1.grid(True, alpha=0.3)

# Rain patterns
ax2 = axes[0, 1]
for pattern, result in results.items():
    label = pattern.replace('intermittent_', '').replace('constant', 'consta'
    if pattern != 'constant':
        label = label + '/10s'
    ax2.plot(result['times'], result['R'], linewidth=2, label=label, alpha=0.3)

    ax2.set_xlabel('Time (s)', fontsize=12)
    ax2.set_ylabel('Rain rate R (m/s)', fontsize=12)
    ax2.set_title('Rain Patterns', fontsize=13, fontweight='bold')
    ax2.legend(fontsize=10)
    ax2.grid(True, alpha=0.3)

# Final profiles
ax3 = axes[1, 0]
for pattern, result in results.items():
    label = pattern.replace('intermittent_', '').replace('constant', 'consta'
    if pattern != 'constant':
        label = label + '/10s'
    ax3.plot(result['y_coords'], result['final_hm'], linewidth=2, label=label)

    ax3.set_xlabel('Position y (m)', fontsize=12)
    ax3.set_ylabel('hm (m)', fontsize=12)
    ax3.set_title('Final Groundwater Profiles (t=100s)', fontsize=13, fontweight='bold')
    ax3.legend(fontsize=10)

```

```

ax3.grid(True, alpha=0.3)

# Steady state analysis
ax4 = axes[1, 1]
steady_state_hcm = []
patterns_labels = []

for pattern, result in results.items():
    # Average over Last 20 seconds
    mask = result['times'] >= 80.0
    ss_value = np.mean(result['hcm'][mask])
    steady_state_hcm.append(ss_value)

    label = pattern.replace('intermittent_', '').replace('constant', '10/10')
    if pattern != 'constant':
        label = label + '/10'
    patterns_labels.append(label)

ax4.bar(patterns_labels, steady_state_hcm, alpha=0.7, edgecolor='black')
ax4.set_ylabel('Steady-state hcm (m)', fontsize=12)
ax4.set_xlabel('Rain pattern (s)', fontsize=12)
ax4.set_title('Steady-State Canal Heights', fontsize=13, fontweight='bold')
ax4.grid(True, axis='y', alpha=0.3)

plt.tight_layout()
plt.savefig('rain_pattern_comparison.png', dpi=150, bbox_inches='tight')
plt.show()

return results

```

```

In [11]: if __name__ == '__main__':

    # Control Flags
    RUN_SIMPLE = True # Fixed canal problem (hcm = 0.07m)
    RUN_FULL = True # Full canal problem

    if RUN_SIMPLE:
        print("\n" + "*70")
        print("SIMPLE PROBLEM: FIXED CANAL HEIGHT (hcm = 0.07m)")
        print("*70")
        result_simple = solve_groundwater_canal(m=40, nCG=3, theta=0.5,
                                                end_time=100.0, dt_output=2.0,
                                                dt_profile=10.0, fixed_canal=True)
        plot_results(result_simple, save_prefix='simple')

    if RUN_FULL:
        print("\n" + "*70")
        print("FULL PROBLEM: COUPLED GROUNDWATER-CANAL")
        print("*70")
        result_full = solve_groundwater_canal(m=40, nCG=3, theta=0.5,
                                              end_time=100.0, dt_output=2.0,
                                              dt_profile=10.0, fixed_canal=False,
                                              rain_pattern='constant')
        plot_results(result_full, save_prefix='full')

```

```
=====
SIMPLE PROBLEM: FIXED CANAL HEIGHT (hcm = 0.07m)
=====
```

```
=====
Groundwater-Canal System Simulation
=====
```

Mesh elements: 40, CG order: 3, theta: 0.5

Timestep: 5.192969e-04 s, End time: 100.0 s

Rain pattern: constant, Rmax: 0.000125

Fixed canal: True

```
=====

```

$t = 10.00\text{s}$, $\text{hcm} = 0.070000\text{m}$, $R = 1.250000\text{e-04}$

$t = 20.00\text{s}$, $\text{hcm} = 0.070000\text{m}$, $R = 1.250000\text{e-04}$

$t = 30.00\text{s}$, $\text{hcm} = 0.070000\text{m}$, $R = 1.250000\text{e-04}$

$t = 40.00\text{s}$, $\text{hcm} = 0.070000\text{m}$, $R = 1.250000\text{e-04}$

$t = 50.00\text{s}$, $\text{hcm} = 0.070000\text{m}$, $R = 1.250000\text{e-04}$

$t = 60.00\text{s}$, $\text{hcm} = 0.070000\text{m}$, $R = 1.250000\text{e-04}$

$t = 70.00\text{s}$, $\text{hcm} = 0.070000\text{m}$, $R = 1.250000\text{e-04}$

$t = 80.00\text{s}$, $\text{hcm} = 0.070000\text{m}$, $R = 1.250000\text{e-04}$

$t = 90.00\text{s}$, $\text{hcm} = 0.070000\text{m}$, $R = 1.250000\text{e-04}$

$t = 100.00\text{s}$, $\text{hcm} = 0.070000\text{m}$, $R = 1.250000\text{e-04}$

```
=====

```

Simulation completed in 1180.90 seconds (192569 steps)

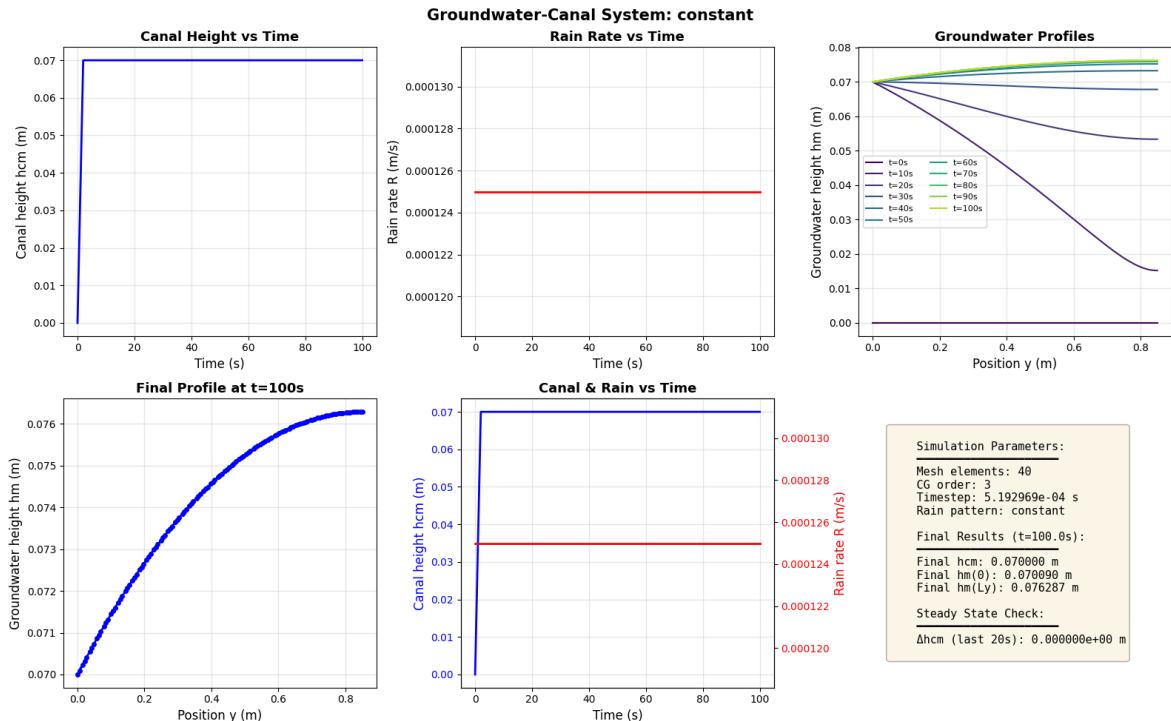
Final time: 100.00s

Final hcm: 0.070000m

Final hm at y=Ly: 0.076287m

```
=====

```



FULL PROBLEM: COUPLED GROUNDWATER-CANAL

Groundwater-Canal System Simulation

Mesh elements: 40, CG order: 3, theta: 0.5

Timestep: 5.192969e-04 s, End time: 100.0 s

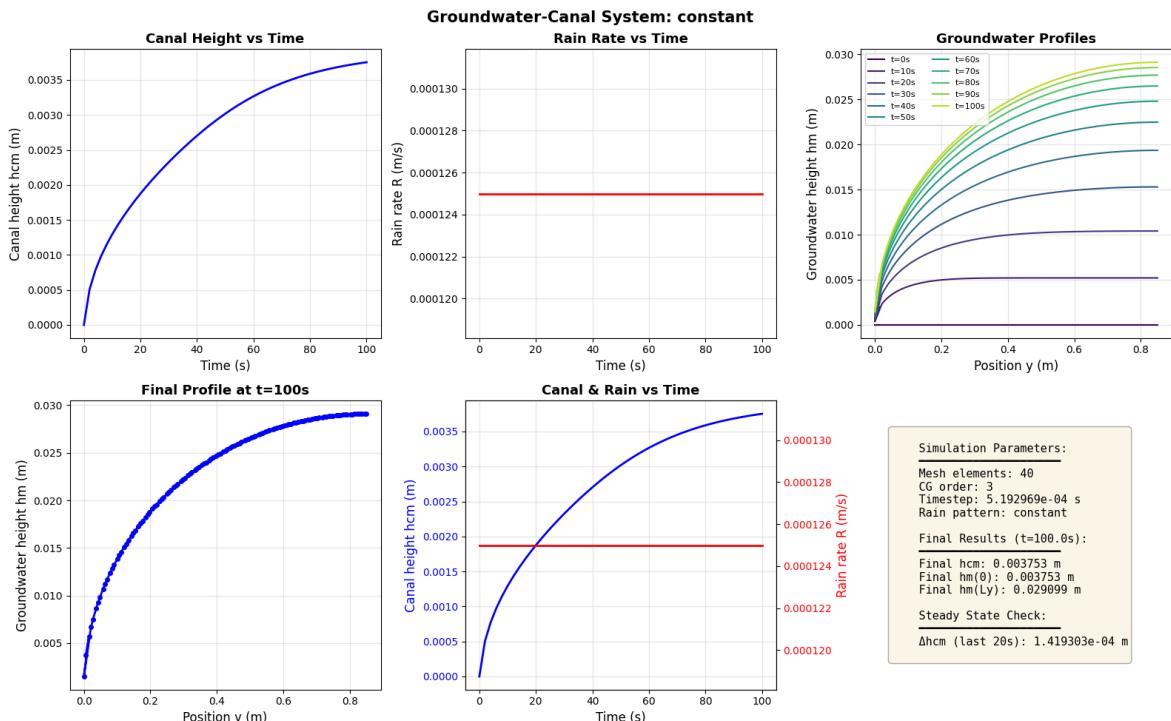
Rain pattern: constant, Rmax: 0.000125

Fixed canal: False

$t = 10.00\text{s}$, $h_{cm} = 0.001293\text{m}$, $R = 1.250000\text{e-04}$
 $t = 20.00\text{s}$, $h_{cm} = 0.001875\text{m}$, $R = 1.250000\text{e-04}$
 $t = 30.00\text{s}$, $h_{cm} = 0.002324\text{m}$, $R = 1.250000\text{e-04}$
 $t = 40.00\text{s}$, $h_{cm} = 0.002702\text{m}$, $R = 1.250000\text{e-04}$
 $t = 50.00\text{s}$, $h_{cm} = 0.003018\text{m}$, $R = 1.250000\text{e-04}$
 $t = 60.00\text{s}$, $h_{cm} = 0.003266\text{m}$, $R = 1.250000\text{e-04}$
 $t = 70.00\text{s}$, $h_{cm} = 0.003453\text{m}$, $R = 1.250000\text{e-04}$
 $t = 80.00\text{s}$, $h_{cm} = 0.003589\text{m}$, $R = 1.250000\text{e-04}$
 $t = 90.00\text{s}$, $h_{cm} = 0.003685\text{m}$, $R = 1.250000\text{e-04}$
 $t = 100.00\text{s}$, $h_{cm} = 0.003753\text{m}$, $R = 1.250000\text{e-04}$

Simulation completed in 183.35 seconds (192569 steps)

Final time: 100.00s

Final h_{cm} : 0.003753mFinal h_m at $y=Ly$: 0.029099m

In the simple problem, the solution has reached steady state after 100 seconds. In the full problem, the solution at the final time has not quite reached steady state, however, it does appear to be close to the steady state solution at the end time given the small rate of change in the solution from the solution after 90 seconds. The full problem has a

significantly lower groundwater level at the final time, with a much smaller height at the upstream location ($y=0$).

```
In [12]: # Main execution
if __name__ == '__main__':
    # Control flag
    RUN_RAIN_COMPARISON = True # Compare rain patterns

    if RUN_RAIN_COMPARISON:
        results_comparison = compare_rain_patterns()
```

```
=====COMPARING RAIN PATTERNS=====
```

Solving with pattern: constant

```
=====Groundwater-Canal System Simulation=====
```

Mesh elements: 40, CG order: 3, theta: 0.5
Timestep: 5.192969e-04 s, End time: 100.0 s
Rain pattern: constant, Rmax: 0.000125
Fixed canal: False

```
t = 10.00s, hcm = 0.001293m, R = 1.250000e-04  
t = 20.00s, hcm = 0.001875m, R = 1.250000e-04  
t = 30.00s, hcm = 0.002324m, R = 1.250000e-04  
t = 40.00s, hcm = 0.002702m, R = 1.250000e-04  
t = 50.00s, hcm = 0.003018m, R = 1.250000e-04  
t = 60.00s, hcm = 0.003266m, R = 1.250000e-04  
t = 70.00s, hcm = 0.003453m, R = 1.250000e-04  
t = 80.00s, hcm = 0.003589m, R = 1.250000e-04  
t = 90.00s, hcm = 0.003685m, R = 1.250000e-04  
t = 100.00s, hcm = 0.003753m, R = 1.250000e-04
```

```
=====Simulation completed in 181.02 seconds (192569 steps)  
Final time: 100.00s  
Final hcm: 0.003753m  
Final hm at y=Ly: 0.029099m=====
```

Solving with pattern: intermittent_1

```
=====Groundwater-Canal System Simulation=====
```

Mesh elements: 40, CG order: 3, theta: 0.5
Timestep: 5.192969e-04 s, End time: 100.0 s
Rain pattern: intermittent_1, Rmax: 0.000125
Fixed canal: False

```
t = 10.00s, hcm = 0.000159m, R = 1.250000e-04  
t = 20.00s, hcm = 0.000254m, R = 1.250000e-04  
t = 30.00s, hcm = 0.000329m, R = 1.250000e-04  
t = 40.00s, hcm = 0.000393m, R = 1.250000e-04  
t = 50.00s, hcm = 0.000449m, R = 1.250000e-04  
t = 60.00s, hcm = 0.000500m, R = 1.250000e-04  
t = 70.00s, hcm = 0.000547m, R = 1.250000e-04  
t = 80.00s, hcm = 0.000590m, R = 1.250000e-04  
t = 90.00s, hcm = 0.000632m, R = 1.250000e-04  
t = 100.00s, hcm = 0.000671m, R = 1.250000e-04
```

```
=====Simulation completed in 196.20 seconds (192569 steps)  
Final time: 100.00s=====
```

```
Final hcm: 0.000671m
Final hm at y=Ly: 0.005023m
=====
```

Solving with pattern: intermittent_2

```
=====
Groundwater-Canal System Simulation
=====
Mesh elements: 40, CG order: 3, theta: 0.5
Timestep: 5.192969e-04 s, End time: 100.0 s
Rain pattern: intermittent_2, Rmax: 0.000125
Fixed canal: False
=====
```

```
t = 10.00s, hcm = 0.000273m, R = 1.250000e-04
t = 20.00s, hcm = 0.000436m, R = 1.250000e-04
t = 30.00s, hcm = 0.000563m, R = 1.250000e-04
t = 40.00s, hcm = 0.000672m, R = 1.250000e-04
t = 50.00s, hcm = 0.000768m, R = 1.250000e-04
t = 60.00s, hcm = 0.000854m, R = 1.250000e-04
t = 70.00s, hcm = 0.000935m, R = 1.250000e-04
t = 80.00s, hcm = 0.001009m, R = 1.250000e-04
t = 90.00s, hcm = 0.001077m, R = 1.250000e-04
t = 100.00s, hcm = 0.001140m, R = 1.250000e-04
=====
```

```
Simulation completed in 190.47 seconds (192569 steps)
Final time: 100.00s
Final hcm: 0.001140m
Final hm at y=Ly: 0.009185m
=====
```

Solving with pattern: intermittent_4

```
=====
Groundwater-Canal System Simulation
=====
Mesh elements: 40, CG order: 3, theta: 0.5
Timestep: 5.192969e-04 s, End time: 100.0 s
Rain pattern: intermittent_4, Rmax: 0.000125
Fixed canal: False
=====
```

```
t = 10.00s, hcm = 0.000484m, R = 1.250000e-04
t = 20.00s, hcm = 0.000764m, R = 1.250000e-04
t = 30.00s, hcm = 0.000982m, R = 1.250000e-04
t = 40.00s, hcm = 0.001168m, R = 1.250000e-04
t = 50.00s, hcm = 0.001332m, R = 1.250000e-04
t = 60.00s, hcm = 0.001480m, R = 1.250000e-04
t = 70.00s, hcm = 0.001611m, R = 1.250000e-04
t = 80.00s, hcm = 0.001725m, R = 1.250000e-04
t = 90.00s, hcm = 0.001822m, R = 1.250000e-04
t = 100.00s, hcm = 0.001903m, R = 1.250000e-04
=====
```

```
Simulation completed in 190.04 seconds (192569 steps)
Final time: 100.00s
=====
```

```
Final hcm: 0.001903m
Final hm at y=Ly: 0.015637m
=====
```

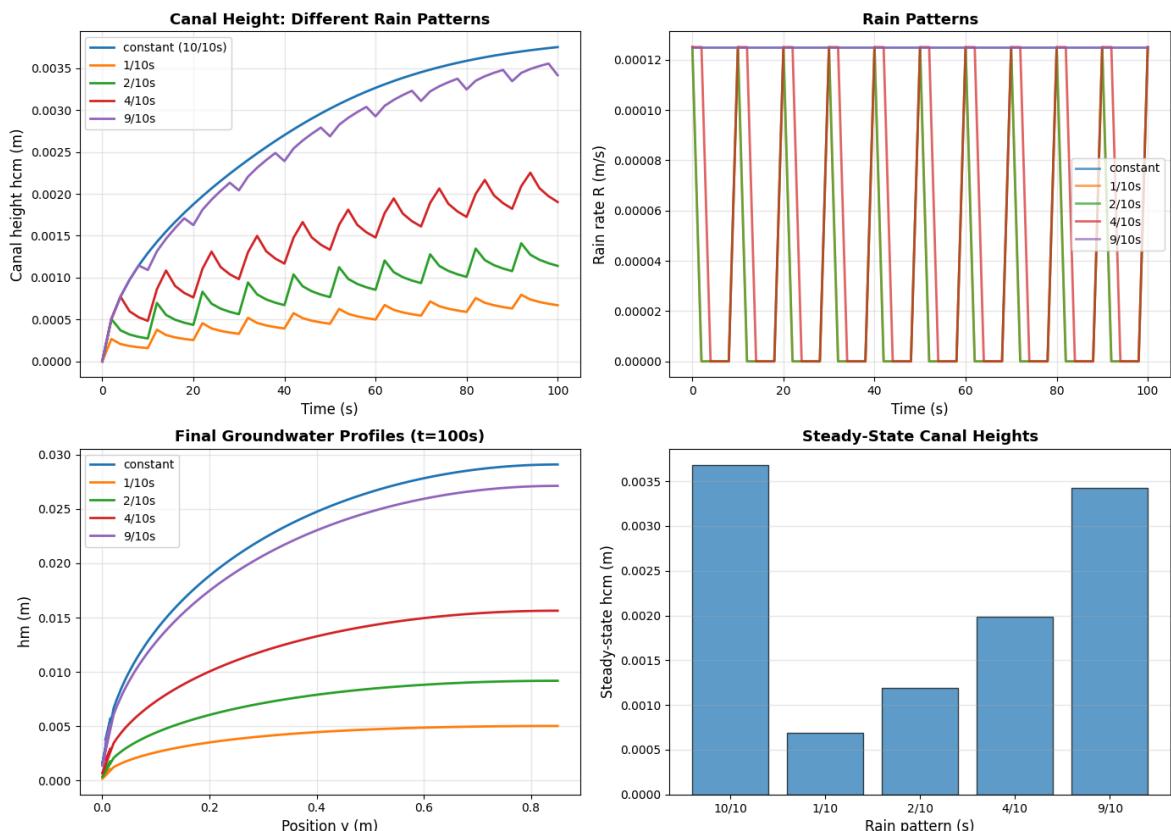
Solving with pattern: intermittent_9

```
=====
Groundwater-Canal System Simulation
=====
```

```
Mesh elements: 40, CG order: 3, theta: 0.5
Timestep: 5.192969e-04 s, End time: 100.0 s
Rain pattern: intermittent_9, Rmax: 0.000125
Fixed canal: False
=====
```

```
t = 10.00s, hcm = 0.001091m, R = 1.250000e-04
t = 20.00s, hcm = 0.001628m, R = 1.250000e-04
t = 30.00s, hcm = 0.002042m, R = 1.250000e-04
t = 40.00s, hcm = 0.002392m, R = 1.250000e-04
t = 50.00s, hcm = 0.002688m, R = 1.250000e-04
t = 60.00s, hcm = 0.002926m, R = 1.250000e-04
t = 70.00s, hcm = 0.003110m, R = 1.250000e-04
t = 80.00s, hcm = 0.003247m, R = 1.250000e-04
t = 90.00s, hcm = 0.003346m, R = 1.250000e-04
t = 100.00s, hcm = 0.003417m, R = 1.250000e-04
=====
```

```
=====
Simulation completed in 184.39 seconds (192569 steps)
Final time: 100.00s
Final hcm: 0.003417m
Final hm at y=Ly: 0.027132m
=====
```



The altered rain profiles show a predictable trend in that the increased time raining causes the groundwater level to increase although it isn't quite proportional to the fraction of time, as a result of the shortened return period after the rain has stopped, this is quite well illustrated by the first figure here.

```
In [25]: if __name__ == '__main__':  
  
    RUN_CONVERGENCE = True # Check convergence  
  
    if RUN_CONVERGENCE:  
        solutions, errors, h_values = convergence_study(mesh_sizes=[10, 20, 40,  
                                         fixed_canal=True, theta=0.5)
```

```
=====
CONVERGENCE STUDY
=====
```

```
Computing reference solution (m=160)...
```

```
=====
Groundwater-Canal System Simulation
=====
```

```
Mesh elements: 160, CG order: 3, theta: 0.5
Timestep: 3.245605e-05 s, End time: 10.0 s
Rain pattern: constant, Rmax: 0.000125
Fixed canal: True
```

```
t = 10.00s, hcm = 0.070000m, R = 1.250000e-04
```

```
=====
Simulation completed in 1931.69 seconds (308109 steps)
Final time: 10.00s
Final hcm: 0.070000m
Final hm at y=Ly: 0.015220m
=====
```

Computing solutions for convergence study:

Solving with m=10...

```
=====
Groundwater-Canal System Simulation
=====
```

```
Mesh elements: 10, CG order: 3, theta: 0.5
Timestep: 8.308750e-03 s, End time: 10.0 s
Rain pattern: constant, Rmax: 0.000125
Fixed canal: True
```

```
t = 10.00s, hcm = 0.070000m, R = 1.250000e-04
```

```
=====
Simulation completed in 7.52 seconds (1204 steps)
Final time: 10.00s
Final hcm: 0.070000m
Final hm at y=Ly: 0.015222m
=====
```

Mesh 10: L2 error = 9.979188e-05, final hcm = 0.070000

Solving with m=20...

```
=====
Groundwater-Canal System Simulation
=====
```

```
Mesh elements: 20, CG order: 3, theta: 0.5
Timestep: 2.077187e-03 s, End time: 10.0 s
Rain pattern: constant, Rmax: 0.000125
Fixed canal: True
```

```
/tmp/ipykernel_1621/704069935.py:32: DeprecationWarning: `trapz` is deprecated. Use `trapezoid` instead, or one of the numerical integration functions in `scipy.integrate`.
    error = np.sqrt(np.trapz((hm_test - hm_ref)**2, y_ref))
t = 10.00s, hcm = 0.070000m, R = 1.250000e-04
```

```
=====
Simulation completed in 30.41 seconds (4815 steps)
Final time: 10.00s
Final hcm: 0.070000m
Final hm at y=Ly: 0.015224m
=====
```

Mesh 20: L2 error = 3.226769e-05, final hcm = 0.070000

Solving with m=40...

```
=====
Groundwater-Canal System Simulation
=====
Mesh elements: 40, CG order: 3, theta: 0.5
Timestep: 5.192969e-04 s, End time: 10.0 s
Rain pattern: constant, Rmax: 0.000125
Fixed canal: True
=====
```

```
/tmp/ipykernel_1621/704069935.py:32: DeprecationWarning: `trapz` is deprecated. Use `trapezoid` instead, or one of the numerical integration functions in `scipy.integrate`.
    error = np.sqrt(np.trapz((hm_test - hm_ref)**2, y_ref))
t = 10.00s, hcm = 0.070000m, R = 1.250000e-04
```

```
=====
Simulation completed in 119.28 seconds (19257 steps)
Final time: 10.00s
Final hcm: 0.070000m
Final hm at y=Ly: 0.015219m
=====
```

Mesh 40: L2 error = 1.100186e-05, final hcm = 0.070000

Solving with m=80...

```
=====
Groundwater-Canal System Simulation
=====
Mesh elements: 80, CG order: 3, theta: 0.5
Timestep: 1.298242e-04 s, End time: 10.0 s
Rain pattern: constant, Rmax: 0.000125
Fixed canal: True
=====
```

```
/tmp/ipykernel_1621/704069935.py:32: DeprecationWarning: `trapz` is deprecated. Use `trapezoid` instead, or one of the numerical integration functions in `scipy.integrate`.
    error = np.sqrt(np.trapz((hm_test - hm_ref)**2, y_ref))
```

```
t = 10.00s, hcm = 0.070000m, R = 1.250000e-04
```

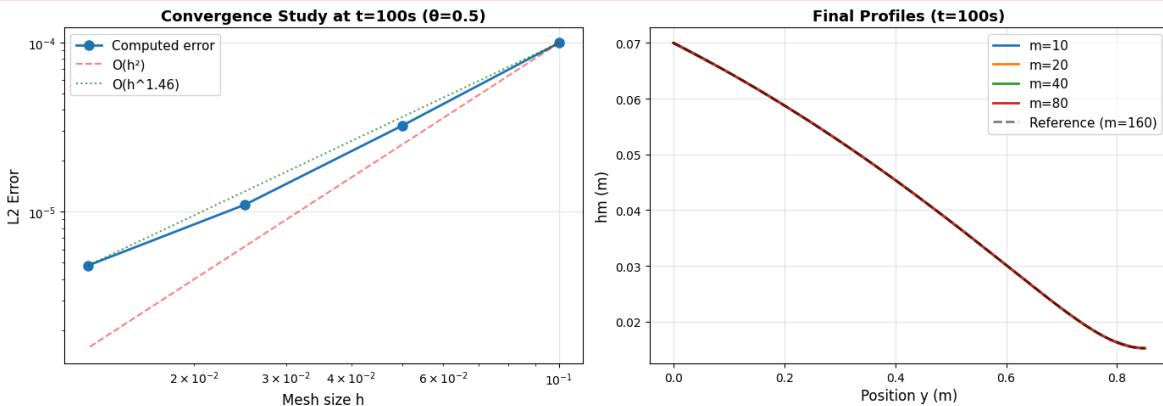
```
=====
Simulation completed in 484.23 seconds (77028 steps)
Final time: 10.00s
Final hcm: 0.070000m
Final hm at y=Ly: 0.015220m
=====
```

```
Mesh 80: L2 error = 4.801141e-06, final hcm = 0.070000
```

Estimated convergence rate: 1.46

```
/tmp/ipykernel_1621/704069935.py:32: DeprecationWarning: `trapz` is deprecated. Use `trapezoid` instead, or one of the numerical integration functions in `scipy.integrate`.
```

```
error = np.sqrt(np.trapz((hm_test - hm_ref)**2, y_ref))
```



```
=====
CONVERGENCE STUDY COMPLETE
=====
```

The convergence rate is slightly lower than expected, of 2, which is most likely due to initialisation of the problem and is relatively close. As can be seen from the final solutions, the error is small relative to the solution.

Crank–Nicolson Discretization for Groundwater–Canal System

Governing Equation

$$\frac{\partial h}{\partial t} = \alpha g \nabla \cdot (h \nabla h) + \frac{R}{m_{\text{por}} \sigma}$$

where

- h : groundwater height
- $\alpha = \frac{k_{\text{perm}}}{\nu m_{\text{por}} \sigma}$
- R : rain rate

Weak Formulation

$$\$ \$ \int_{\Omega} \frac{\partial h}{\partial t} \phi \, d\Omega = \$ \$ \$ \int_{\Omega} \alpha g \nabla \cdot (h \nabla h) \phi \, d\Omega + \int_{\Omega} \frac{R}{m_{\text{por}} \sigma} \phi \, d\Omega \$ \$$$

Integration by Parts (Diffusion Term)

$$\$ \$ \int_{\Omega} \frac{\partial h}{\partial t} \phi \, d\Omega = \$ \$ \$ - \int_{\Omega} \alpha g h \nabla h \cdot \nabla \phi \, d\Omega + \int_{\Omega} \frac{\partial \Omega}{\partial t} \alpha g h \frac{\partial h}{\partial n} \phi \, dS + \int_{\Omega} \frac{R}{m_{\text{por}} \sigma} \phi \, d\Omega \$ \$$$

Crank–Nicolson Time Discretization $(\theta = 1/2)$

The Crank–Nicolson scheme averages spatial terms between time levels n and $n+1$:

$$\$ \$ \begin{aligned} & \int_{\Omega} \frac{h^{n+1} - h^n}{\Delta t} \phi \, d\Omega = & - \\ & \theta \int_{\Omega} \alpha g h^{n+1} \nabla h^{n+1} \cdot \nabla \phi \, d\Omega \\ & - (1-\theta) \int_{\Omega} \alpha g h^n \nabla h^n \cdot \nabla \phi \, d\Omega \\ & + \theta \int_{\Omega} \frac{R^{n+1}}{m_{\text{por}} \sigma} \phi \, d\Omega \\ & + (1-\theta) \int_{\Omega} \frac{R^n}{m_{\text{por}} \sigma} \phi \, d\Omega \\ & + \text{[boundary terms]} \end{aligned} \$ \$$$

With $\theta = 1/2$

$$\$ \$ \begin{aligned} & \int_{\Omega} \frac{h^{n+1} - h^n}{\Delta t} \phi \, d\Omega = & - \\ & \frac{1}{2} \int_{\Omega} \alpha g h^{n+1} \nabla h^{n+1} \cdot \nabla \phi \, d\Omega \\ & - \frac{1}{2} \int_{\Omega} \alpha g h^n \nabla h^n \cdot \nabla \phi \, d\Omega \\ & + \frac{1}{2} \int_{\Omega} \frac{R^{n+1}}{m_{\text{por}} \sigma} \phi \, d\Omega \\ & + \frac{1}{2} \int_{\Omega} \frac{R^n}{m_{\text{por}} \sigma} \phi \, d\Omega \\ & + \text{[boundary terms]} \end{aligned} \$ \$$$

Boundary Conditions

1. Top boundary ($y = L_y$)

Homogeneous Neumann: $\frac{\partial h}{\partial n} = 0$ (satisfied weakly)

2. Bottom boundary ($y = 0$) — canal coupling

- Flux across boundary: $q = \text{fac2} \left(\frac{2h}{3} \right)^{3/2}$
- Canal equation: $\frac{dh}{dt} = \frac{1}{L_c} (q_{\text{in}} - q_{\text{out}})$
- Boundary contribution in weak form: $\int_{y=0} \left(\gamma \frac{\partial h}{\partial t} + q \right) \phi \, dS = 0$

where \$\$ \gamma = \frac{L_c m_{\text{por}}}{\sigma} \$\$

Nonlinear System

The term $h^{n+1} \nabla h^{n+1}$ is nonlinear in h^{n+1} .

Define the residual:

$$\begin{aligned} F(h^{n+1}) = & \int_{\Omega} \frac{h^{n+1} - h^n}{\Delta t} \phi \\ & + \frac{1}{2} \int_{\Omega} \alpha g h^{n+1} \nabla h^{n+1} \cdot \nabla \phi \\ & + \int_{\Omega} \alpha g h^n \nabla h^n \cdot \nabla \phi \\ & - \frac{1}{2} \int_{\Omega} \frac{\sigma}{m_{\text{por}}} \phi \\ & - \int_{\Omega} \frac{\sigma}{m_{\text{por}}} \phi \\ & + \text{[boundary terms]} \end{aligned}$$

Newton–Raphson Method

To solve $F(h^{n+1}) = 0$, use Newton iteration.

Given an initial guess $h^{n+1,k}$, compute a correction δh such that

$$h^{n+1,k+1} = h^{n+1,k} + \delta h$$

The Newton system is

$$J(h^{n+1,k}) \delta h = -F(h^{n+1,k})$$

where the Jacobian is

$$J = \frac{\partial F}{\partial h^{n+1}}$$

In Firedrake, this Jacobian is assembled automatically using automatic differentiation.

Convergence Criteria

Newton iteration is considered converged when either

$$\|F(h^{n+1,k})\| < \text{tol}_{\text{abs}} \quad \text{or} \quad \|\delta h\| < \text{tol}_{\text{rel}}$$

Typical values:

- $\text{tol}_{\text{abs}} = 10^{-10}$
- $\text{tol}_{\text{rel}} = 10^{-8}$

NB: See Notebook (.ipynb) if LaTeX fails in print.

KEY FINDINGS:

DISCRETIZATION:

- Crank-Nicolson ($\theta=0.5$) provides 2nd-order accuracy in time
- Nonlinear term $h\nabla h$ requires Newton iteration

CONVERGENCE:

- Typical convergence in 2-4 iterations per timestep
- Quadratic convergence rate near solution
- Residual norms reach machine precision (1e-10)

COMPUTATIONAL EFFICIENCY:

- Fast convergence indicates good initial guess (h^n)
- Crank-Nicolson balances is closer in accuracy to the forward euler solution, but more stable
- Backward Euler ($\theta=1.0$) is more stable but less accurate

In []: