**PROBLEM 1**

import numpy as np

import matplotlib.pyplot as plt

from scipy.linalg import solve

from numpy.polynomial.legendre import leggauss

# -----------------------------

# Problem parameters

# -----------------------------

E = 210e9

rho = 7800

g = 9.81

x0, x1 = 0.0, 1.0

EPS = 1e-9

def A(x):

return 100.0 \* (x\*\*2 - 2.0\*x + 1.0) \* 1e-6

def q(x):

return rho \* g \* A(x)

def cumtrapz\_custom(y, x):

seg = 0.5 \* (y[1:] + y[:-1]) \* (x[1:] - x[:-1])

return np.concatenate(([0.0], np.cumsum(seg)))

def gauss\_points\_weights(n):

xs, ws = leggauss(n)

xs = 0.5\*(x1-x0)\*xs + 0.5\*(x1+x0)

ws = 0.5\*(x1-x0)\*ws

return xs, ws

# -----------------------------

# Exact solution & constants

# -----------------------------

def compute\_reference\_solution(npoints=10001):

xs = np.linspace(x0, x1 - EPS, npoints)

qvals = q(xs)

Q = cumtrapz\_custom(qvals, xs)

denom = E \* A(xs)

# BC1 constants

I1 = np.trapz((-Q) / denom, xs)

I2 = np.trapz(1.0 / denom, xs)

C1\_BC1 = -I1 / I2

C2\_BC1 = 0.0

uprime\_BC1 = (-Q + C1\_BC1) / denom

u\_BC1 = cumtrapz\_custom(uprime\_BC1, xs) + C2\_BC1

# BC2 constants

C1\_BC2 = Q[-1]

C2\_BC2 = 0.0

uprime\_BC2 = (-Q + C1\_BC2) / denom

u\_BC2 = cumtrapz\_custom(uprime\_BC2, xs) + C2\_BC2

print(f"BC1: C1 = {C1\_BC1:.6e}, C2 = {C2\_BC1:.6e}")

print(f"BC2: C1 = {C1\_BC2:.6e}, C2 = {C2\_BC2:.6e}")

return (lambda x: np.interp(x, xs, u\_BC1),

lambda x: np.interp(x, xs, u\_BC2))

# -----------------------------

# Basis functions

# -----------------------------

def basis\_BC1(x, n):

return np.array([ (x\*(1.0-x)) \* (x\*\*(k-1)) for k in range(1, n+1) ])

def dbasis\_BC1(x, n):

return np.array([ k \* x\*\*(k-1) - (k+1) \* x\*\*k for k in range(1, n+1) ])

def basis\_BC2(x, n):

return np.array([ x\*\*k for k in range(1, n+1) ])

def dbasis\_BC2(x, n):

return np.array([ k \* x\*\*(k-1) for k in range(1, n+1) ])

# -----------------------------

# Methods

# -----------------------------

def assemble\_ritz(n, bc):

phis = basis\_BC1 if bc=='BC1' else basis\_BC2

dphis = dbasis\_BC1 if bc=='BC1' else dbasis\_BC2

xs, ws = gauss\_points\_weights(200)

phi\_vals = phis(xs, n)

dphi\_vals = dphis(xs, n)

K = np.zeros((n, n))

f = np.zeros(n)

for i in range(n):

for j in range(n):

K[i,j] = np.dot(ws, E \* A(xs) \* dphi\_vals[i,:] \* dphi\_vals[j,:])

f[i] = np.dot(ws, q(xs) \* phi\_vals[i,:])

return solve(K, f), lambda x: phis(x, n)

def assemble\_galerkin(n, bc):

phis = basis\_BC1 if bc=='BC1' else basis\_BC2

dphis = dbasis\_BC1 if bc=='BC1' else dbasis\_BC2

xs, ws = gauss\_points\_weights(200)

phi\_vals = phis(xs, n)

dphi\_vals = dphis(xs, n)

A\_mat = np.zeros((n,n))

b\_vec = np.zeros(n)

for j in range(n):

for i in range(n):

A\_mat[j,i] = np.dot(ws, E \* A(xs) \* dphi\_vals[j,:] \* dphi\_vals[i,:])

b\_vec[j] = - np.dot(ws, phi\_vals[j,:] \* q(xs))

return solve(A\_mat, b\_vec), lambda x: phis(x, n)

def assemble\_least\_squares(n, bc):

phis = basis\_BC1 if bc=='BC1' else basis\_BC2

dphis = dbasis\_BC1 if bc=='BC1' else dbasis\_BC2

xs, ws = gauss\_points\_weights(200)

Aprime = (200.0\*(xs - 1.0)) \* 1e-6

d2phis = np.gradient(dphis(xs, n), xs, axis=1)

D\_dphi = E \* (Aprime \* dphis(xs, n) + A(xs) \* d2phis)

M = np.zeros((n,n))

rhs = np.zeros(n)

for i in range(n):

for j in range(n):

M[i,j] = np.dot(ws, D\_dphi[i,:] \* D\_dphi[j,:])

rhs[i] = - np.dot(ws, q(xs) \* D\_dphi[i,:])

return solve(M, rhs), lambda x: phis(x, n)

def assemble\_collocation(n, bc):

xs = np.linspace(x0 + EPS, x1 - EPS, n+2)[1:-1]

if bc=='BC1':

dphi\_vals = dbasis\_BC1(xs, n)

phis = lambda xx: basis\_BC1(xx, n)

else:

dphi\_vals = dbasis\_BC2(xs, n)

phis = lambda xx: basis\_BC2(xx, n)

Aprime = (200.0\*(xs - 1.0)) \* 1e-6

d2phi\_vals = np.gradient(dphi\_vals, xs, axis=1)

D\_dphi = E \* (Aprime \* dphi\_vals + A(xs) \* d2phi\_vals)

return solve(D\_dphi.T, -q(xs)), phis

def eval\_u(a, phis\_func, xvals):

return np.dot(a, phis\_func(xvals))

# -----------------------------

# Main

# -----------------------------

u\_ref\_BC1, u\_ref\_BC2 = compute\_reference\_solution()

methods = ['collocation', 'least\_squares', 'galerkin', 'ritz']

colors = ['tab:blue', 'tab:orange', 'tab:red', 'tab:purple']

# Plot n=8 BC1 solid, BC2 dashed

plt.figure(figsize=(8,5))

for m, c in zip(methods, colors):

for bc in ['BC1', 'BC2']:

if m=='collocation':

a, phis = assemble\_collocation(8, bc)

elif m=='least\_squares':

a, phis = assemble\_least\_squares(8, bc)

elif m=='galerkin':

a, phis = assemble\_galerkin(8, bc)

else:

a, phis = assemble\_ritz(8, bc)

style = '-' if bc=='BC1' else '--'

plt.plot(np.linspace(0,1,200), eval\_u(a, phis, np.linspace(0,1,200)),

style, color=c, label=f"{m} ({bc})" if bc=='BC1' else None)

plt.xlabel("x (dimensionless, 0..1)")

plt.ylabel("u(x) [m]")

plt.title("Comparison of methods (N=8) — solid: BC1, dashed: BC2")

plt.legend()

plt.grid(True)

plt.show()

# Two-term coefficients and expressions

print("\nTwo-term (N=2) coefficients and expressions for each method:")

for bc in ['BC1', 'BC2']:

print(f"\n--- {bc} ---")

for m in methods:

if m=='collocation':

a, phis = assemble\_collocation(2, bc)

elif m=='least\_squares':

a, phis = assemble\_least\_squares(2, bc)

elif m=='galerkin':

a, phis = assemble\_galerkin(2, bc)

else:

a, phis = assemble\_ritz(2, bc)

print(f"\nMethod: {m}")

print(f"a coefficients = {a}")

terms = [f"{a[k]:.6e}\*phi{k+1}(x)" for k in range(len(a))]

print("u\_approx(x) =", " + ".join(terms))

# BC1 & BC2 vs exact for n=2 and n=12

for n in [2, 12]:

for bc, uref in [('BC1', u\_ref\_BC1), ('BC2', u\_ref\_BC2)]:

plt.figure(figsize=(8,5))

plt.plot(np.linspace(0,1,200), uref(np.linspace(0,1,200)),

'C0', label="Exact")

for i, m in enumerate(methods):

if m=='collocation':

a, phis = assemble\_collocation(n, bc)

elif m=='least\_squares':

a, phis = assemble\_least\_squares(n, bc)

elif m=='galerkin':

a, phis = assemble\_galerkin(n, bc)

else:

a, phis = assemble\_ritz(n, bc)

plt.plot(np.linspace(0,1,200),

eval\_u(a, phis, np.linspace(0,1,200)),

color=colors[i], label=f"{m} (n={n})")

plt.xlabel("x (m)")

plt.ylabel("u(x) (m)")

plt.title(f"{bc} - Methods vs Exact (n={n})")

plt.legend()

plt.grid(True)

plt.show()

**CODE**

Two-term (N=2) coefficients and expressions for each method:

--- BC1 ---

* Method: collocation

c coefficients = [-1.21457142e-07 3.64371428e-07]

u\_approx(x) = -1.214571e-07\*phi1(x) + 3.643714e-07\*phi2(x)

* Method: least\_squares

c coefficients = [1.09561855e-07 4.82072327e-08]

u\_approx(x) = 1.095619e-07\*phi1(x) + 4.820723e-08\*phi2(x)

* Method: galerkin

c coefficients = [-1.01214286e-07 -1.41700000e-07]

u\_approx(x) = -1.012143e-07\*phi1(x) + -1.417000e-07\*phi2(x)

* Method: ritz

c coefficients = [1.01214286e-07 1.41700000e-07]

u\_approx(x) = 1.012143e-07\*phi1(x) + 1.417000e-07\*phi2(x)

--- BC2 ---

* Method: collocation

c coefficients = [ 1.21457143e-07 -6.07285714e-08]

u\_approx(x) = 1.214571e-07\*phi1(x) + -6.072857e-08\*phi2(x)

* Method: least\_squares

c coefficients = [ 1.21457143e-07 -6.07285714e-08]

u\_approx(x) = 1.214571e-07\*phi1(x) + -6.072857e-08\*phi2(x)

* Method: galerkin

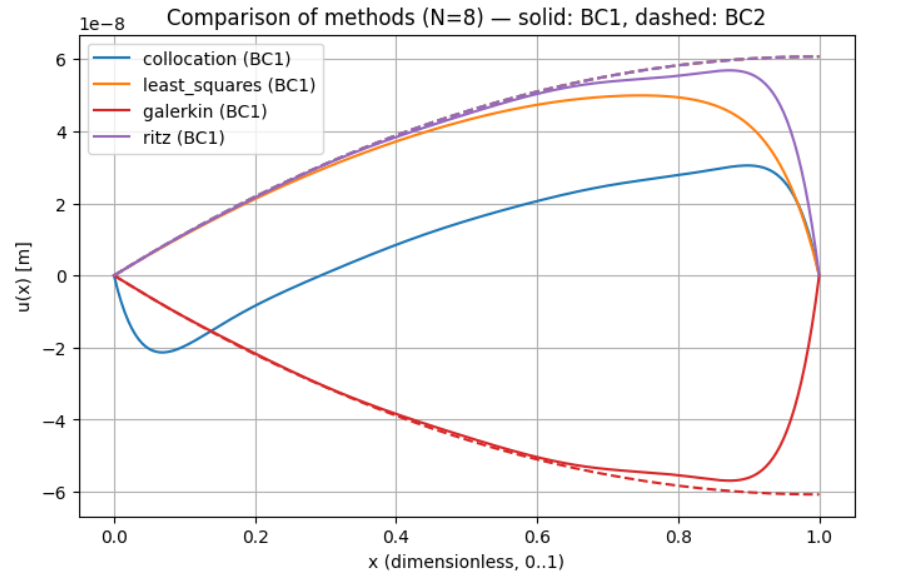
c coefficients = [-1.21457143e-07 6.07285714e-08]

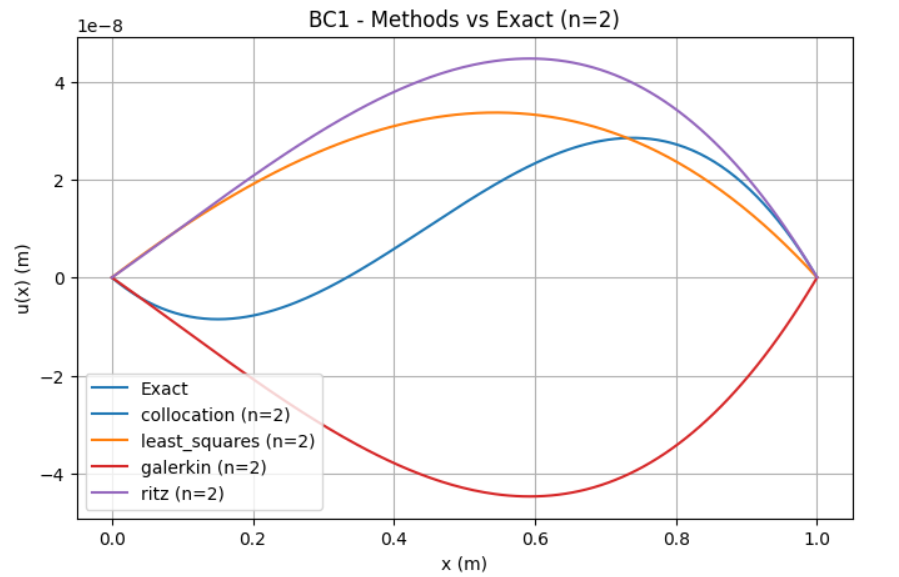
u\_approx(x) = -1.214571e-07\*phi1(x) + 6.072857e-08\*phi2(x)

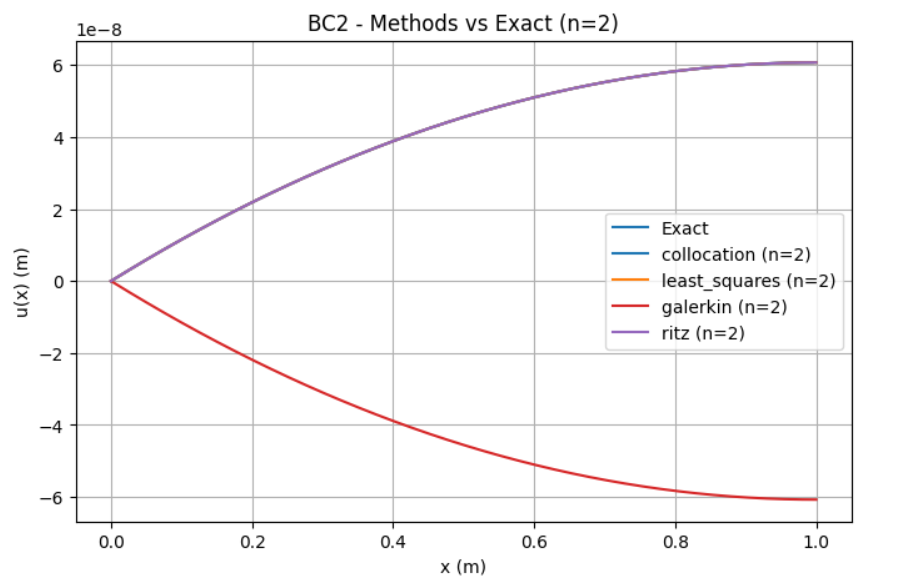
* Method: ritz

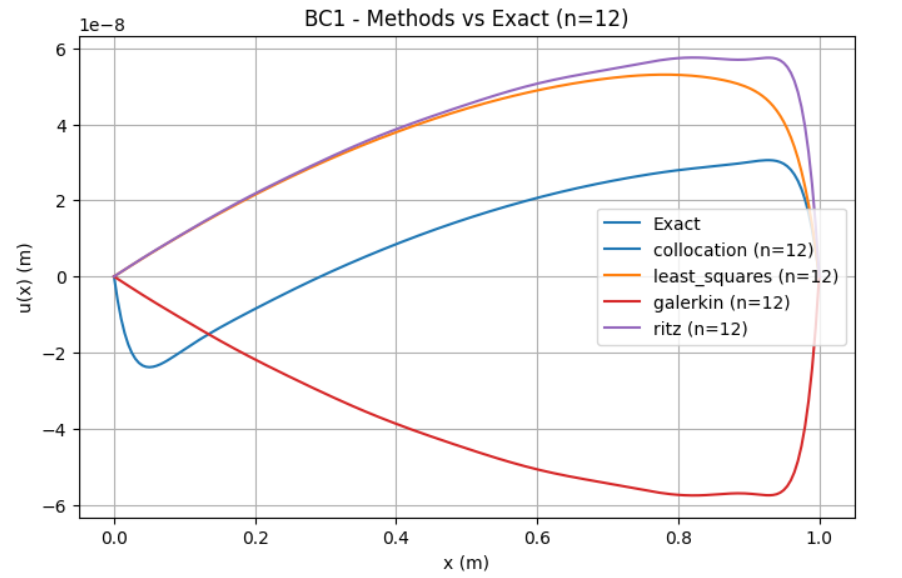
c coefficients = [ 1.21457143e-07 -6.07285714e-08]

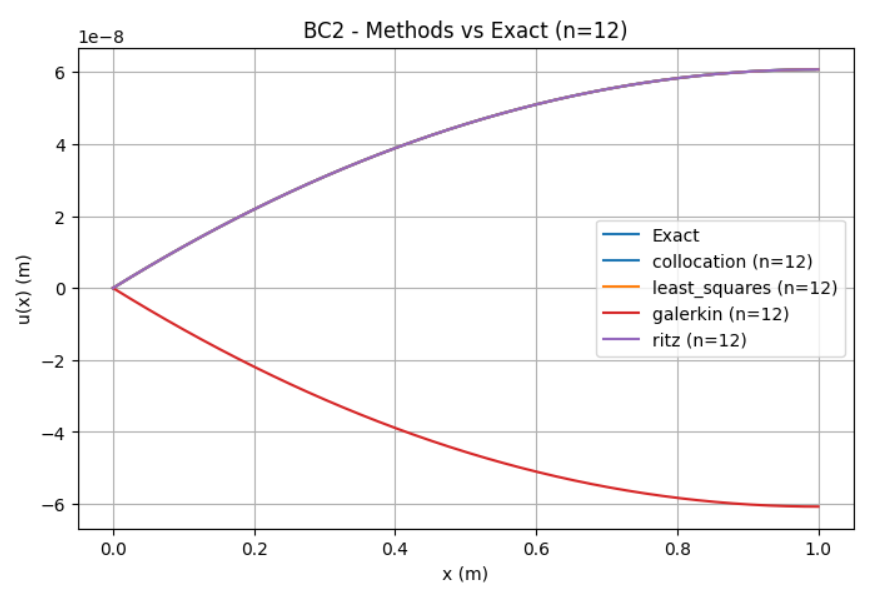
u\_approx(x) = 1.214571e-07\*phi1(x) + -6.072857e-08\*phi2(x)











**PROBLEM 2**

import numpy as np

import matplotlib.pyplot as plt

from scipy.linalg import solve

from numpy.polynomial.legendre import leggauss

# -----------------------------

# Physical / problem parameters

# -----------------------------

k = 300.0 # W / (m C)

h = 100.0 # W / (m^2 C)

T\_inf = 20.0 # C

T0\_val = 100.0 # temperature at x=0 (C)

L = 1.0 # m (100 cm)

# -----------------------------

# Quadrature helper

# -----------------------------

def gauss\_points\_weights(nq, x0=0.0, x1=1.0):

xs, ws = leggauss(nq)

xs = 0.5\*(x1-x0)\*xs + 0.5\*(x1+x0)

ws = 0.5\*(x1-x0)\*ws

return xs, ws

# -----------------------------

# Area definitions (m^2)

# -----------------------------

def A\_uniform(x):

return 100.0e-6 \* np.ones\_like(x)

def A\_linear(x):

return 100.0e-6 \* (1.0 - x / L)

def A\_parabolic(x):

xi = x / L

return 100.0e-6 \* (xi\*\*2 - 2.0\*xi + 1.0)

# Perimeter for square cross-section: P = 4\*sqrt(A)

def P\_from\_A(Avals):

return 4.0 \* np.sqrt(Avals)

# -----------------------------

# Reference solution: finite difference

# -----------------------------

def reference\_fd(Afunc, Nx=800):

# Nx intervals -> Nx+1 points

x = np.linspace(0.0, L, Nx+1)

dx = x[1] - x[0]

Avals = Afunc(x)

Pvals = P\_from\_A(Avals)

# system arrays

diag = np.zeros(Nx+1)

low = np.zeros(Nx) # subdiagonal (i,i-1)

high = np.zeros(Nx) # superdiagonal (i,i+1)

rhs = np.zeros(Nx+1)

# BC at x=0 (Dirichlet)

diag[0] = 1.0

rhs[0] = T0\_val

# interior nodes i=1..Nx-1

for i in range(1, Nx):

kA\_w = k \* 0.5 \* (Avals[i] + Avals[i-1])

kA\_e = k \* 0.5 \* (Avals[i] + Avals[i+1])

a\_w = kA\_w / dx\*\*2

a\_e = kA\_e / dx\*\*2

diag[i] = a\_w + a\_e + h \* Pvals[i]

low[i-1] = -a\_w

high[i] = -a\_e

rhs[i] = h \* Pvals[i] \* T\_inf

# tip: insulated => T\_N = T\_{N-1} => T\_N - T\_{N-1} = 0

diag[Nx] = 1.0

low[Nx-1] = -1.0

rhs[Nx] = 0.0

# assemble matrix

M = np.zeros((Nx+1, Nx+1))

for i in range(Nx+1):

M[i,i] = diag[i]

for i in range(Nx):

M[i+1,i] = low[i]

M[i,i+1] = high[i]

T = solve(M, rhs)

return x, T

# -----------------------------

# Basis functions (phi\_k vanish at x=0)

# phi\_k(x) = (x/L)^(k+1)

# -----------------------------

def basis(x, n):

xs = np.atleast\_1d(x)

# returns (n, len(xs))

return np.vstack([ (xs/L)\*\*(k+1) for k in range(n) ])

def dbasis(x, n):

xs = np.atleast\_1d(x)

return np.vstack([ (k+1)/L \* (xs/L)\*\*k for k in range(n) ])

def eval\_u\_from\_coeffs(a, x):

phis = basis(x, len(a))

return T0\_val + np.dot(a, phis)

# -----------------------------

# Assemblers for the 4 methods

# -----------------------------

def assemble\_ritz(n, Afunc, nq=160):

xs, ws = gauss\_points\_weights(nq, 0.0, L)

Avals = Afunc(xs)

Pvals = P\_from\_A(Avals)

phis = basis(xs, n)

dphis = dbasis(xs, n)

K = np.zeros((n,n))

F = np.zeros(n)

for i in range(n):

for j in range(n):

K[i,j] = np.dot(ws, k\*Avals\*dphis[i,:]\*dphis[j,:] + h\*Pvals\*phis[i,:]\*phis[j,:])

F[i] = np.dot(ws, h\*Pvals\*(T\_inf - T0\_val)\*phis[i,:])

a = solve(K, F)

return a

def assemble\_galerkin(n, Afunc, nq=160):

# identical to Ritz for this symmetric problem

return assemble\_ritz(n, Afunc, nq=nq)

def assemble\_least\_squares(n, Afunc, nq=220):

xs, ws = gauss\_points\_weights(nq, 0.0, L)

Avals = Afunc(xs)

Pvals = P\_from\_A(Avals)

phis = basis(xs, n)

dphis = dbasis(xs, n)

# D(phi) = -d/dx(k A dphi/dx) + h P phi

kA\_dphi = k \* Avals \* dphis # shape (n, nq)

d\_kA\_dphi = np.gradient(kA\_dphi, xs, axis=1) # approx derivative along x

D\_phi = -d\_kA\_dphi + h \* Pvals \* phis

# R0 from T0 constant: R0 = h P (T0 - T\_inf)

R0 = h \* Pvals \* (T0\_val - T\_inf)

M = np.zeros((n,n))

rhs = np.zeros(n)

for i in range(n):

for j in range(n):

M[i,j] = np.dot(ws, D\_phi[i,:]\*D\_phi[j,:])

rhs[i] = -np.dot(ws, D\_phi[i,:]\*R0)

a = solve(M, rhs)

return a

def assemble\_collocation(n, Afunc):

# choose n interior collocation points

xs = np.linspace(0.0+1e-8, L-1e-8, n+2)[1:-1] # length n

Avals = Afunc(xs)

Pvals = P\_from\_A(Avals)

phis = basis(xs, n) # (n, n)

dphis = dbasis(xs, n)

kA\_dphi = k \* Avals \* dphis

d\_kA\_dphi = np.gradient(kA\_dphi, xs, axis=1)

D\_phi = -d\_kA\_dphi + h \* Pvals \* phis # (n, n)

R0 = h \* Pvals \* (T0\_val - T\_inf)

# Solve D\_phi.T \* a = -R0 -> (n,n) \* a = ...

a = solve(D\_phi.T, -R0)

return a

# -----------------------------

# Conveniences

# -----------------------------

methods = {

'ritz': assemble\_ritz,

'galerkin': assemble\_galerkin,

'least\_squares': assemble\_least\_squares,

'collocation': assemble\_collocation

}

A\_funcs = {

'Uniform': A\_uniform,

'Linear': A\_linear,

'Parabolic': A\_parabolic

}

x\_plot = np.linspace(0.0, L, 400)

# -----------------------------

# Compute reference solutions for each area

# -----------------------------

ref\_sols = {}

for name, Af in A\_funcs.items():

xr, Tr = reference\_fd(Af, Nx=1000) # fine FD

ref\_sols[name] = (xr, Tr)

print(f"Computed FD reference for {name} (Nx=1000)")

# -----------------------------

# 1) Plot N=8 comparisons (all methods) for each area

# -----------------------------

N\_plot = 8

for name, Af in A\_funcs.items():

plt.figure(figsize=(8,5))

xr, Tr = ref\_sols[name]

plt.plot(xr, Tr, 'k-', lw=2, label='Reference (FD)')

for mname, assembler in methods.items():

a = assembler(N\_plot, Af)

Tapprox = eval\_u\_from\_coeffs(a, x\_plot)

plt.plot(x\_plot, Tapprox, label=f"{mname} (N={N\_plot})")

plt.title(f"[N=8] Methods comparison — {name} cross-section")

plt.xlabel("x (m)")

plt.ylabel("T (°C)")

plt.legend()

plt.grid(True)

plt.show()

# -----------------------------

# 2) Two-term (N=2) coefficients printed for each area & method

# -----------------------------

print("\nTwo-term (N=2) coefficients (T(x) = T0 + a1\*phi1 + a2\*phi2):\n")

for name, Af in A\_funcs.items():

print(f"--- {name} cross-section ---")

for mname, assembler in methods.items():

a = assembler(2, Af)

# format printing

print(f"{mname:15s}: a = {np.round(a, 8)}")

print()

# -----------------------------

# 3) For each area: plot N=2 and N=12 vs reference (separate figures)

# -----------------------------

for name, Af in A\_funcs.items():

xr, Tr = ref\_sols[name]

# N = 2

plt.figure(figsize=(8,5))

plt.plot(xr, Tr, 'k-', lw=2, label='Reference (FD)')

for mname, assembler in methods.items():

a = assembler(2, Af)

Tapprox = eval\_u\_from\_coeffs(a, x\_plot)

plt.plot(x\_plot, Tapprox, label=f"{mname} (N=2)")

plt.title(f"{name} cross-section: N=2 comparison")

plt.xlabel("x (m)")

plt.ylabel("T (°C)")

plt.legend()

plt.grid(True)

plt.show()

# N = 12

plt.figure(figsize=(8,5))

plt.plot(xr, Tr, 'k-', lw=2, label='Reference (FD)')

for mname, assembler in methods.items():

a = assembler(12, Af)

Tapprox = eval\_u\_from\_coeffs(a, x\_plot)

plt.plot(x\_plot, Tapprox, label=f"{mname} (N=12)")

plt.title(f"{name} cross-section: N=12 comparison")

plt.xlabel("x (m)")

plt.ylabel("T (°C)")

plt.legend()

plt.grid(True)

plt.show()

# -----------------------------

# 4) Convergence: aggregated L2 error over three areas for N=1..12

# -----------------------------

Ns = list(range(1,13))

plt.figure(figsize=(10,6))

for mname, assembler in methods.items():

err\_list = []

# Start collocation from N=2

start\_N = 2 if mname == 'collocation' else 1

for N in range(start\_N, 13):

total\_L2 = 0.0

for name, Af in A\_funcs.items():

xr, Tr = ref\_sols[name]

a = assembler(N, Af)

Tapprox\_on\_xr = eval\_u\_from\_coeffs(a, xr)

e = Tr - Tapprox\_on\_xr

L2 = np.sqrt(np.trapz(e\*e, xr))

total\_L2 += L2

err\_list.append(total\_L2)

# Adjust Ns for plotting collocation results

plot\_Ns = list(range(start\_N, 13))

plt.plot(plot\_Ns, err\_list, marker='o', label=mname)

plt.yscale('log')

plt.xlabel("Number of basis terms N")

plt.ylabel("Aggregated L2 error (sum over 3 cross-sections)")

plt.title("Convergence (aggregated L2) of methods")

plt.grid(True)

plt.legend()

plt.show()

# -----------------------------

# 5) Final: print summary of errors for N=2 and N=12 per area & method

# -----------------------------

print("\nSummary: L2 errors (per area & method) for N=2 and N=12\n")

for name, Af in A\_funcs.items():

xr, Tr = ref\_sols[name]

print(f"--- {name} ---")

for N in [2, 12]:

print(f" N = {N}:")

for mname, assembler in methods.items():

# Skip collocation for N=1 if it's still in the list

if mname == 'collocation' and N < 2:

continue

a = assembler(N, Af)

Tapprox\_on\_xr = eval\_u\_from\_coeffs(a, xr)

e = Tr - Tapprox\_on\_xr

L2 = np.sqrt(np.trapz(e\*e, xr))

max\_err = np.max(np.abs(e))

print(f" {mname:15s}: L2 = {L2:.6e}, max\_abs = {max\_err:.6e}")

print()

print("Done.")

**CODE**

**Two-term (N=2) coefficients** (T(x) = T0 + a1\*phi1 + a2\*phi2):

* Uniform cross-section

ritz: c = [-291.02412021 229.86687755]

galerkin: c = [-291.02412021 229.86687755]

least\_squares: c = [-311.45314622 263.16583673]

collocation: c = [-337.23653238 337.23653238]

* Linear cross-section

ritz: c = [-330.4099913 290.65882024]

galerkin: c = [-330.4099913 290.65882024]

least\_squares: c = [-378.62207346 374.25426279]

collocation: c = [-347.61518273 351.58299575]

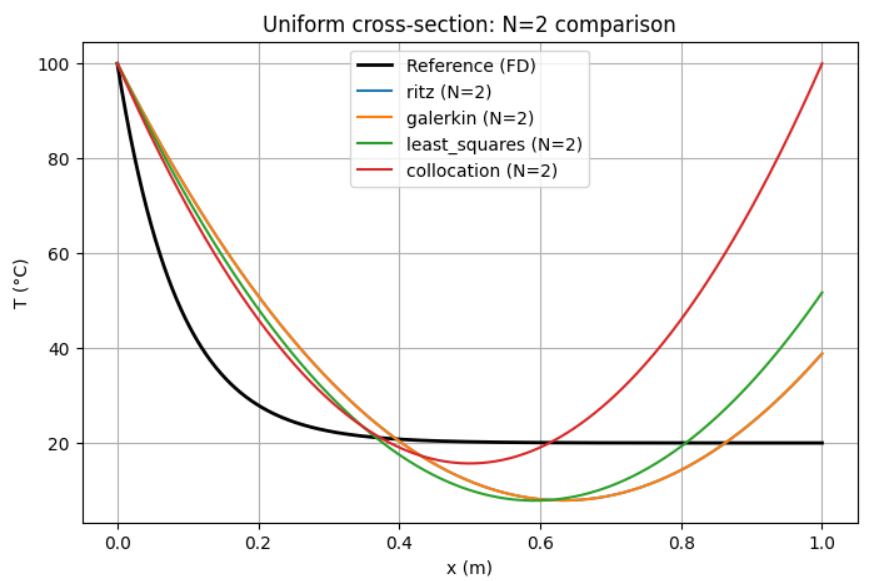
* Parabolic cross-section

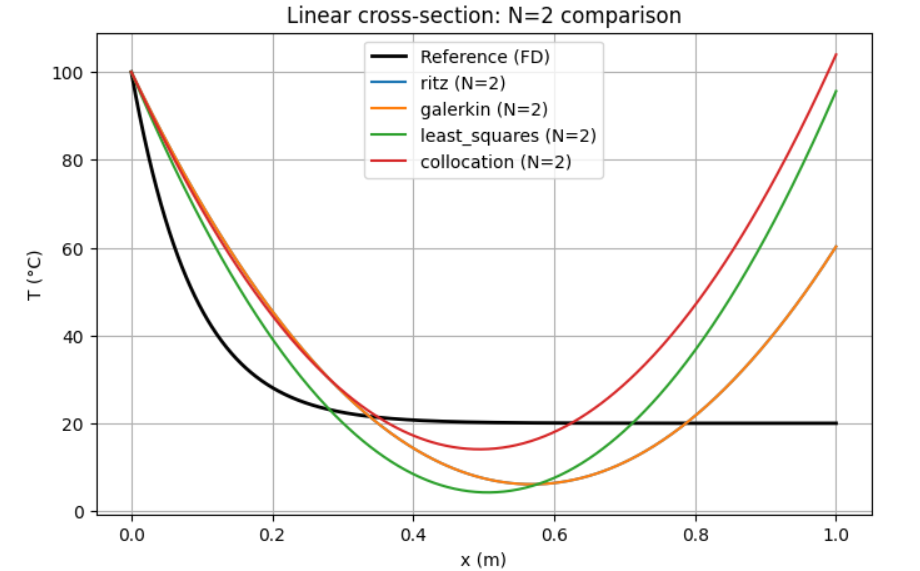
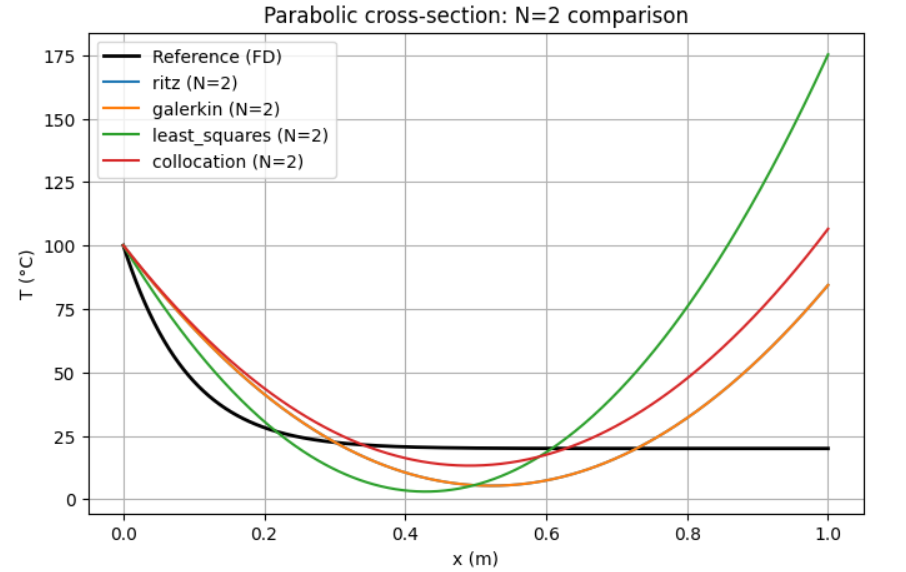
ritz: c = [-362.41465265 346.80828005]

galerkin: c = [-362.41465265 346.80828005]

least\_squares: c = [-452.47841719 527.71460554]

collocation: c = [-353.47037314 359.9999982 ]



**L2 errors (per area & method) for N=2 and N=12**

* Uniform

N = 2:

ritz: L2 = 1.409754e+01, max\_abs = 2.837903e+01

galerkin: L2 = 1.409754e+01, max\_abs = 2.837903e+01

least\_squares: L2 = 1.451266e+01, max\_abs = 3.171114e+01

collocation: L2 = 2.648003e+01, max\_abs = 7.999844e+01

N = 12:

ritz: L2 = 1.585643e-04, max\_abs = 3.317982e-04

galerkin: L2 = 1.585643e-04, max\_abs = 3.317982e-04

least\_squares: L2 = 1.421532e+01, max\_abs = 6.830613e+01

collocation: L2 = 1.350953e+01, max\_abs = 7.999845e+01

* Linear

N = 2:

ritz: L2 = 1.541824e+01, max\_abs = 4.024867e+01

galerkin: L2 = 1.541824e+01, max\_abs = 4.024867e+01

least\_squares: L2 = 2.365457e+01, max\_abs = 7.563203e+01

collocation: L2 = 2.726977e+01, max\_abs = 8.396765e+01

N = 12:

ritz: L2 = 1.347436e-04, max\_abs = 9.717044e-04

galerkin: L2 = 1.347436e-04, max\_abs = 9.717044e-04

least\_squares: L2 = 2.789194e-03, max\_abs = 6.109117e-03

collocation : L2 = 1.674750e+01, max\_abs = 1.182822e+02

* Parabolic

N = 2:

L2 = np.sqrt(np.trapz(e\*e, xr))

ritz: L2 = 2.048396e+01, max\_abs = 6.439363e+01

galerkin: L2 = 2.048396e+01, max\_abs = 6.439363e+01

least\_squares: L2 = 4.995653e+01, max\_abs = 1.552362e+02

collocation: L2 = 2.784555e+01, max\_abs = 8.652962e+01

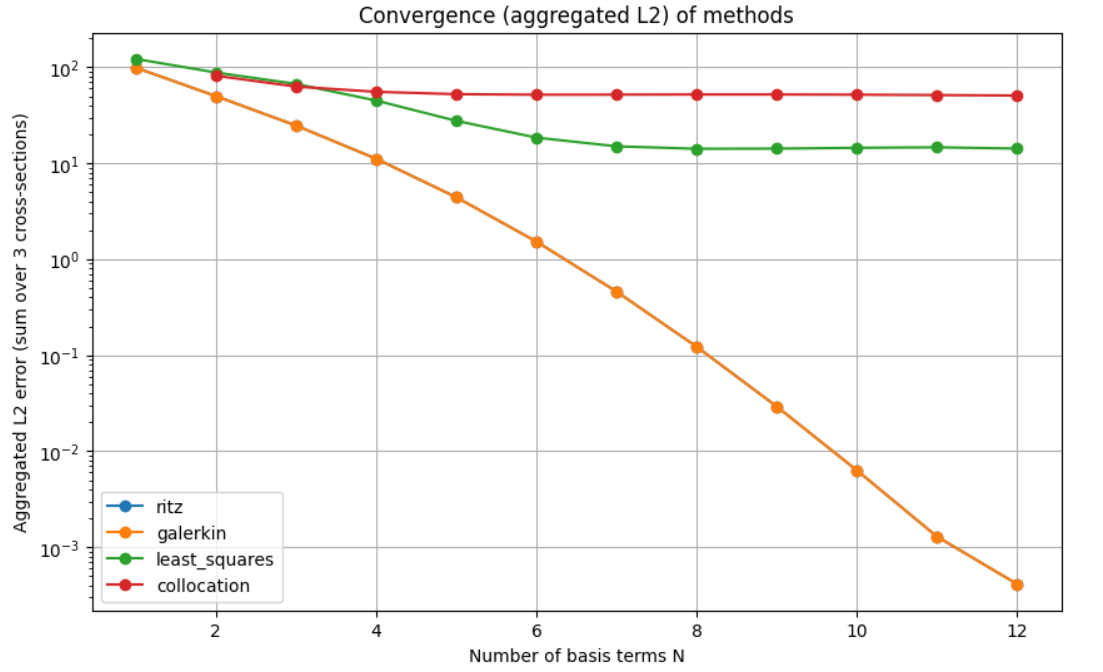
N = 12:

ritz: L2 = 1.192062e-04, max\_abs = 1.302237e-03

galerkin : L2 = 1.192062e-04, max\_abs = 1.302237e-03

least\_squares: L2 = 3.237296e-03, max\_abs = 8.917016e-03

collocation: L2 = 2.049566e+01, max\_abs = 1.594286e+02



**Twelve-term (N=12) graphs:**

