TOPOLOGY OPTIMIZATION

MAE 598 DESIGN OPTIMIZATTION

MONISH DEV SUDHAKHAR

1220859588

Topology Optimization

Problem Formulation:

A MBB is a classical problem in which The design domain, the boundary conditions, and the external load for the MBB beam are shown in Fig. 1 problem is to find the optimal material distribution, in terms of minimum compliance, with a constraint on the total amount of material.

Approach:

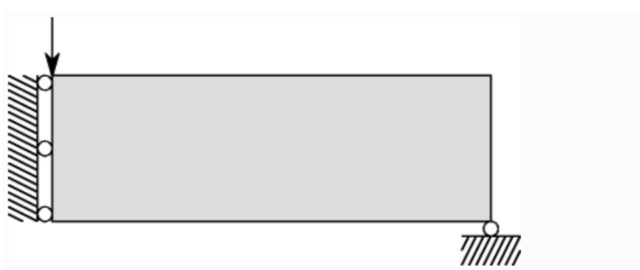
The design domain is discretized by square finite elements and a “density-based approach to topology optimization” is followed (Bendsøe 1989; Zhou and Rozvany 1991); i.e. each element *e* is assigned a density x_e that determines its Young’s modulus E_e:

$$Ee(xe) = Emin + xpe(E0 - Emin), xe \in [0,1]$$

where E_0 is the stiffness of the material, $E_{\min}$ is a very small stiffness assigned to void regions in order to prevent the stiffness matrix from becoming singular, and p is a penalization factor (typically $p = 3$) introduced to ensure black-and-white solutions.

The compliance minimization problem

Topology optimization has been commonly used to design structures and materials with optimal mechanical, thermal, electromagnetic, acoustical, or other properties. The structure under design is segmented into n finite elements, and a density value xi is assigned to each element i∈{1,2,...,n}: A higher density corresponds to a less porous material element and higher Yong's modulus. Reducing the density to zero is equivalent to creating a hole in the structure. Thus, the set of densities x={xi} can be used to represent the topology of the structure and is considered as the variables to be optimized. A common topology optimization problem is compliance minimization, where we seek the "stiffest" structure within a certain volume limit to withhold a particular load:



$$\text{subject to:} \quad \mathbf{h} := \mathbf{K}(\mathbf{x})\mathbf{d} = \mathbf{u},$$

$$\mathbf{g} := V(\mathbf{x}) \leq v,$$

$$\mathbf{x} \in [0, 1].$$

Here V(x) is the total volume; v is an upper bound on volume; d∈Rnd×1 is the displacement of the structure under the load u, where nd is the degrees of freedom (DOF) of the system (i.e., the number of x- and y-coordinates of nodes from the finite element model of the structure); K(x) is the global stiffness matrix for the structure.

K(x) is indirectly influenced by the topology x, through the element-wise stiffness matrix

$$\mathbf{K}_i = \bar{\mathbf{K}}_e E(x_i),$$

$$\mathbf{K}(\mathbf{x}) = \mathbf{G}[\mathbf{K}_1, \mathbf{K}_2, \ldots, \mathbf{K}_n],$$

where the matrix K¯e is predefined according to the finite element type (we use first-order quadrilateral elements throughout this tutorial) and the nodal displacements of the element (we use square elements with unit lengths throughout this tutorial), G is an assembly matrix, E(xi) is the element-wise Young's modulus defined as a function of the density xi: E(xi):=ΔExip+Emin, where p (the penalty parameter) is usually set to 3. This cubic relationship between the topology and the modulus is determined by the material constitutive models, and numerically, it also helps binarize the topologies, i.e., to push the optimal xi to 1 or 0 (why?). The term Emin is added to provide numerical stability.

Design sensitivity analysis

This problem has both inequality and equality constraints. However, the inequality ones are only related to the topology x, and are either linear (V(x)≤v) or simple bounds (x∈[0,1]). We will show that these constraints can be easily handled. The problem thus involves two sets of variables: We can consider x as the **decision variables** and u as the state variables that are governed by x through the equality constraint K(x)d=u.

The reduced gradient (often called design sensitivity) can be calculated as

$$\frac{df}{d\mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}} - \frac{\partial f}{\partial \mathbf{u}}\left(\frac{\partial \mathbf{h}}{\partial \mathbf{u}}\right)^{-1}\frac{\partial \mathbf{h}}{\partial \mathbf{x}},$$

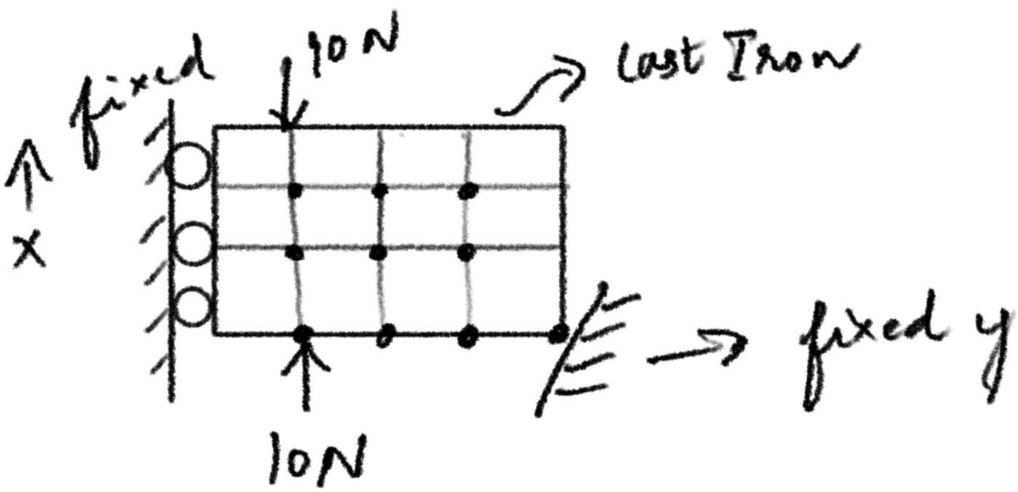
$$\frac{df}{d\mathbf{x}} = -\mathbf{u}^T\frac{\partial \mathbf{K}}{\partial \mathbf{x}}\mathbf{u}.$$

Recall the relation between K and x, and notice that

$$\mathbf{u}^T\mathbf{K}\mathbf{u} = \sum_{i=1}^{n} \mathbf{u}_i^T\mathbf{K}_i\mathbf{u}_i,$$

Problem Formulation:

The system is a beam with Cast iron as the material. It is subjected to a compressive load of 10 N. The material properties are changed with respect to the material and the boundary



CODE IN PYTHON

```
# Monish Dev Sudhakhar
from __future__ import division
import numpy as np

from scipy.sparse import coo_matrix
from scipy.sparse.linalg import spsolve
from matplotlib import colors
import matplotlib.pyplot as plt
import cvxopt;
import cvxopt.cholmod


def main(nelx, nely, volfrac, penal, rmin, ft):
    print("Minimum compliance problem with OC")
    print("ndes: " + str(nelx) + " x " + str(nely))
    print("volfrac: " + str(volfrac) + ", rmin: " + str(rmin) + ", penal: " + str(penal))
    print("Filter method: " + ["Sensitivity based", "Density based"][ft])

    # Max and min stiffness (Cast iron is being considered)
    Emin = 1e-9
    Emax = 1.0
```

```python
# dofs:
ndof = 2 * (nelx + 1) * (nely + 1)

# Allocate design variables (as array), initialize and allocate sens.
x = volfrac * np.ones(nely * nelx, dtype=float)
xold = x.copy()
xPhys = x.copy()

g = 0  # must be initialized to use the NGuyen/Paulino OC approach
dc = np.zeros((nely, nelx), dtype=float)

# FE: Build the index vectors for the for coo matrix format.
KE = lk()
edofMat = np.zeros((nelx * nely, 8), dtype=int)
for elx in range(nelx):
    for ely in range(nely):
        el = ely + elx * nely
        n1 = (nely + 1) * elx + ely
        n2 = (nely + 1) * (elx + 1) + ely
        edofMat[el, :] = np.array(
            [2 * n1 + 2, 2 * n1 + 3, 2 * n2 + 2, 2 * n2 + 3, 2 * n2, 2 * n2 + 1, 2 * n1, 2 * n1 + 1])
# Construct the index pointers for the coo format
iK = np.kron(edofMat, np.ones((8, 1))).flatten()
jK = np.kron(edofMat, np.ones((1, 8))).flatten()

# Filter: Build (and assemble) the index+data vectors for the coo matrix format
nfilter = int(nelx * nely * ((2 * (np.ceil(rmin) - 1) + 1) ** 2))
iH = np.zeros(nfilter)
jH = np.zeros(nfilter)
sH = np.zeros(nfilter)
cc = 0
for i in range(nelx):
    for j in range(nely):
        row = i * nely + j
        kk1 = int(np.maximum(i - (np.ceil(rmin) - 1), 0))
        kk2 = int(np.minimum(i + np.ceil(rmin), nelx))
        ll1 = int(np.maximum(j - (np.ceil(rmin) - 1), 0))
        ll2 = int(np.minimum(j + np.ceil(rmin), nely))
        for k in range(kk1, kk2):
            for l in range(ll1, ll2):
                col = k * nely + l
                fac = rmin - np.sqrt(((i - k) * (i - k) + (j - l) * (j - l)))
                iH[cc] = row
                jH[cc] = col
                sH[cc] = np.maximum(0.0, fac)
                cc = cc + 1
# Finalize assembly and convert to csc format
H = coo_matrix((sH, (iH, jH)), shape=(nelx * nely, nelx * nely)).tocsc()
Hs = H.sum(1)

# BC's and support
dofs = np.arange(2 * (nelx + 1) * (nely + 1))
fixed = np.union1d(dofs[0:2 * (nely + 1):2], np.array([2 * (nelx + 1) * (nely + 1) - 1]))
free = np.setdiff1d(dofs, fixed)

# Solution and RHS vectors
f = np.zeros((ndof, 1))
u = np.zeros((ndof, 1))
```

```python
# Set load
f[1, 0] = - 10
# Initialize plot and plot the initial design
plt.ion()  # Ensure that redrawing is possible
fig, ax = plt.subplots()
im = ax.imshow(-xPhys.reshape((nelx, nely)).T, cmap='gray', \
               interpolation='none', norm=colors.Normalize(vmin=-1, vmax=0))
fig.show()

loop = 0
change = 1
dv = np.ones(nely * nelx)
dc = np.ones(nely * nelx)
ce = np.ones(nely * nelx)
while change > 0.0001 and loop < 100:
    loop = loop + 1
    # Setup and solve FE problem
    sK = ((KE.flatten()[np.newaxis]).T * (Emin + (xPhys) ** penal * (Emax - Emin))).flatten(order='F')
    K = coo_matrix((sK, (iK, jK)), shape=(ndof, ndof)).tocsc()
    # Remove constrained dofs from matrix and convert to coo
    K = deleterowcol(K, fixed, fixed).tocoo()
    # Solve system
    K = cvxopt.spmatrix(K.data, K.row.astype(int), K.col.astype(int))
    B = cvxopt.matrix(f[free, 0])
    cvxopt.cholmod.linsolve(K, B)
    u[free, 0] = np.array(B)[:, 0]

    # Objective and sensitivity
    ce[:] = (np.dot(u[edofMat].reshape(nelx * nely, 8), KE) * u[edofMat].reshape(nelx * nely, 8)).sum(1)
    obj = ((Emin + xPhys ** penal * (Emax - Emin)) * ce).sum()
    dc[:] = (-penal * xPhys ** (penal - 1) * (Emax - Emin)) * ce

    dv[:] = np.ones(nely * nelx)
    # Sensitivity filtering:
    if ft == 0:
        dc[:] = np.asarray((H * (x * dc))[np.newaxis].T / Hs)[:, 0] / np.maximum(0.001, x)
    elif ft == 1:
        dc[:] = np.asarray(H * (dc[np.newaxis].T / Hs))[:, 0]
        dv[:] = np.asarray(H * (dv[np.newaxis].T / Hs))[:, 0]

    # Optimality criteria
    xold[:] = x
    (x[:], g) = oc(nelx, nely, x, volfrac, dc, dv, g)

    # Filter design variables
    if ft == 0:
        xPhys[:] = x
    elif ft == 1:
        xPhys[:] = np.asarray(H * x[np.newaxis].T / Hs)[:, 0]

    # Compute the change by the inf. norm
    change = np.linalg.norm(x.reshape(nelx * nely, 1) - xold.reshape(nelx * nely, 1), np.inf)

    # Plot to screen
    im.set_array(-xPhys.reshape((nelx, nely)).T)
    fig.canvas.draw()

    # Write iteration history to screen (req. Python 2.6 or newer)
    print("it.: {0} , obj.: {1:.3f} Vol.: {2:.3f}, ch.: {3:.3f}".format( \
        loop, obj, (g + volfrac * nelx * nely) / (nelx * nely), change))
```

```
    # Make sure the plot stays and that the shell remains
    plt.show()
    input("Press any key...")


# element stiffness matrix (Titanium is being considered)
def lk():
    E = 9.6
    nu = 0.21
    k = np.array(
        [1 / 2 - nu / 6, 1 / 8 + nu / 8, -1 / 4 - nu / 12, -1 / 8 + 3 * nu / 8, -1 / 4 + nu / 12, -1 / 8 - nu / 8,
         nu / 6, 1 / 8 - 3 * nu / 8])
    KE = E / (1 - nu ** 2) * np.array([[k[0], k[1], k[2], k[3], k[4], k[5], k[6], k[7]],
                                       [k[1], k[0], k[7], k[6], k[5], k[4], k[3], k[2]],
                                       [k[2], k[7], k[0], k[5], k[6], k[3], k[4], k[1]],
                                       [k[3], k[6], k[5], k[0], k[7], k[2], k[1], k[4]],
                                       [k[4], k[5], k[6], k[7], k[0], k[1], k[2], k[3]],
                                       [k[5], k[4], k[3], k[2], k[1], k[0], k[7], k[6]],
                                       [k[6], k[3], k[4], k[1], k[2], k[7], k[0], k[5]],
                                       [k[7], k[2], k[1], k[4], k[3], k[6], k[5], k[0]]]);
    return (KE)


def oc(nelx, nely, x, volfrac, dc, dv, g):
    l1 = 0
    l2 = 1e9
    move = 0.2
    # reshape to perform vector operations
    xnew = np.zeros(nelx * nely)

    while (l2 - l1) / (l1 + l2) > 1e-3:
        lmid = 0.5 * (l2 + l1)
        xnew[:] = np.maximum(0.0,
                             np.maximum(x - move, np.minimum(1.0, np.minimum(x + move, x * np.sqrt(-dc / dv / lmid)))))
        gt = g + np.sum((dv * (xnew - x)))
        if gt > 0:
            l1 = lmid
        else:
            l2 = lmid
    return (xnew, gt)


def deleterowcol(A, delrow, delcol):
    # Assumes that matrix is in symmetric csc form !
    m = A.shape[0]
    keep = np.delete(np.arange(0, m), delrow)
    A = A[keep, :]
    keep = np.delete(np.arange(0, m), delcol)
    A = A[:, keep]
    return A


if __name__ == "__main__":
    # Default input parameters
    nelx = 190
    nely = 35
    volfrac = 0.39
    rmin = 3.5
    penal = 2.9
```

```
    ft = 1  # ft==0 -> sens, ft==1 -> dens

    import sys

    if len(sys.argv) > 1: nelx = int(sys.argv[1])
    if len(sys.argv) > 2: nely = int(sys.argv[2])
    if len(sys.argv) > 3: volfrac = float(sys.argv[3])
    if len(sys.argv) > 4: rmin = float(sys.argv[4])
    if len(sys.argv) > 5: penal = float(sys.argv[5])
    if len(sys.argv) > 6: ft = int(sys.argv[6])

main(nelx, nely, volfrac,
```

OPTIMIZATION RESULTS:

C:\Anaconda\python.exe "C:/Users/Monish Dev Sudhakhar/AppData/Roaming/JetBrains/PyCharmCE2021.2/scratches/scratch_4.py"

Minimum compliance problem with OC

ndes: 190 x 35

volfrac: 0.39, rmin: 3.5, penal: 2.9

Filter method: Density based

it.: 1 , obj.: 106344.188 Vol.: 0.390, ch.: 0.200

it.: 2 , obj.: 48799.330 Vol.: 0.390, ch.: 0.200

it.: 3 , obj.: 31953.044 Vol.: 0.390, ch.: 0.200

it.: 4 , obj.: 24227.188 Vol.: 0.390, ch.: 0.200

it.: 5 , obj.: 21879.368 Vol.: 0.390, ch.: 0.200

it.: 6 , obj.: 21191.599 Vol.: 0.390, ch.: 0.200

it.: 7 , obj.: 20801.539 Vol.: 0.390, ch.: 0.124

it.: 8 , obj.: 20530.824 Vol.: 0.390, ch.: 0.195

it.: 9 , obj.: 20301.349 Vol.: 0.390, ch.: 0.115

it.: 10 , obj.: 20068.117 Vol.: 0.390, ch.: 0.168

it.: 11 , obj.: 19818.626 Vol.: 0.390, ch.: 0.168

it.: 12 , obj.: 19632.302 Vol.: 0.390, ch.: 0.178

it.: 13 , obj.: 19465.959 Vol.: 0.390, ch.: 0.137

it.: 14 , obj.: 19322.729 Vol.: 0.390, ch.: 0.147

it.: 15 , obj.: 19198.597 Vol.: 0.390, ch.: 0.091

it.: 16 , obj.: 19085.209 Vol.: 0.390, ch.: 0.116

it.: 17 , obj.: 18972.875 Vol.: 0.390, ch.: 0.136

it.: 18 , obj.: 18866.127 Vol.: 0.390, ch.: 0.092

it.: 19 , obj.: 18776.052 Vol.: 0.390, ch.: 0.090

it.: 20 , obj.: 18677.585 Vol.: 0.390, ch.: 0.095

it.: 21 , obj.: 18587.191 Vol.: 0.390, ch.: 0.103

it.: 22 , obj.: 18478.783 Vol.: 0.390, ch.: 0.100

it.: 23 , obj.: 18366.708 Vol.: 0.390, ch.: 0.112

it.: 24 , obj.: 18257.161 Vol.: 0.390, ch.: 0.130

it.: 25 , obj.: 18147.180 Vol.: 0.390, ch.: 0.126

it.: 26 , obj.: 18031.116 Vol.: 0.390, ch.: 0.151

it.: 27 , obj.: 17914.208 Vol.: 0.390, ch.: 0.142

it.: 28 , obj.: 17799.731 Vol.: 0.390, ch.: 0.182

it.: 29 , obj.: 17672.490 Vol.: 0.390, ch.: 0.200

it.: 30 , obj.: 17548.813 Vol.: 0.390, ch.: 0.200

it.: 31 , obj.: 17372.277 Vol.: 0.390, ch.: 0.200

it.: 32 , obj.: 17171.020 Vol.: 0.390, ch.: 0.200

it.: 33 , obj.: 16962.219 Vol.: 0.390, ch.: 0.200

it.: 34 , obj.: 16765.713 Vol.: 0.390, ch.: 0.194

it.: 35 , obj.: 16614.619 Vol.: 0.390, ch.: 0.200

it.: 36 , obj.: 16497.502 Vol.: 0.390, ch.: 0.200

it.: 37 , obj.: 16381.186 Vol.: 0.390, ch.: 0.187

it.: 38 , obj.: 16279.287 Vol.: 0.390, ch.: 0.164

it.: 39 , obj.: 16185.334 Vol.: 0.390, ch.: 0.136

it.: 40 , obj.: 16081.524 Vol.: 0.390, ch.: 0.147

it.: 41 , obj.: 15973.592 Vol.: 0.390, ch.: 0.171

it.: 42 , obj.: 15877.034 Vol.: 0.390, ch.: 0.165

it.: 43 , obj.: 15775.292 Vol.: 0.390, ch.: 0.170

it.: 44 , obj.: 15675.182 Vol.: 0.390, ch.: 0.133

it.: 45 , obj.: 15578.813 Vol.: 0.390, ch.: 0.162

it.: 46 , obj.: 15494.902 Vol.: 0.390, ch.: 0.133

it.: 47 , obj.: 15412.317 Vol.: 0.390, ch.: 0.138

it.: 48 , obj.: 15333.532 Vol.: 0.390, ch.: 0.139

it.: 49 , obj.: 15257.122 Vol.: 0.390, ch.: 0.158

it.: 50 , obj.: 15177.750 Vol.: 0.390, ch.: 0.160

it.: 51 , obj.: 15105.335 Vol.: 0.390, ch.: 0.173

it.: 52 , obj.: 15021.557 Vol.: 0.390, ch.: 0.200

it.: 53 , obj.: 14941.176 Vol.: 0.390, ch.: 0.164

it.: 54 , obj.: 14863.179 Vol.: 0.390, ch.: 0.153

it.: 55 , obj.: 14782.575 Vol.: 0.390, ch.: 0.162

it.: 56 , obj.: 14698.012 Vol.: 0.390, ch.: 0.172

it.: 57 , obj.: 14610.660 Vol.: 0.390, ch.: 0.179

it.: 58 , obj.: 14525.952 Vol.: 0.390, ch.: 0.178

it.: 59 , obj.: 14462.759 Vol.: 0.390, ch.: 0.154

it.: 60 , obj.: 14423.017 Vol.: 0.390, ch.: 0.156

it.: 61 , obj.: 14393.686 Vol.: 0.390, ch.: 0.145

it.: 62 , obj.: 14367.352 Vol.: 0.390, ch.: 0.139

it.: 63 , obj.: 14347.288 Vol.: 0.390, ch.: 0.121

it.: 64 , obj.: 14326.632 Vol.: 0.390, ch.: 0.117

it.: 65 , obj.: 14309.968 Vol.: 0.390, ch.: 0.123

it.: 66 , obj.: 14295.631 Vol.: 0.390, ch.: 0.132

it.: 67 , obj.: 14278.794 Vol.: 0.390, ch.: 0.144

it.: 68 , obj.: 14266.240 Vol.: 0.390, ch.: 0.182

it.: 69 , obj.: 14250.242 Vol.: 0.390, ch.: 0.128

it.: 70 , obj.: 14237.508 Vol.: 0.390, ch.: 0.161

it.: 71 , obj.: 14225.731 Vol.: 0.390, ch.: 0.126

it.: 72 , obj.: 14212.170 Vol.: 0.390, ch.: 0.124

it.: 73 , obj.: 14202.021 Vol.: 0.390, ch.: 0.098

it.: 74 , obj.: 14192.491 Vol.: 0.390, ch.: 0.082

it.: 75 , obj.: 14184.970 Vol.: 0.390, ch.: 0.083

it.: 76 , obj.: 14180.003 Vol.: 0.390, ch.: 0.084

it.: 77 , obj.: 14176.230 Vol.: 0.390, ch.: 0.083

it.: 78 , obj.: 14173.049 Vol.: 0.390, ch.: 0.080

it.: 79 , obj.: 14169.122 Vol.: 0.390, ch.: 0.072

it.: 80 , obj.: 14166.232 Vol.: 0.390, ch.: 0.072

it.: 81 , obj.: 14164.416 Vol.: 0.390, ch.: 0.072

it.: 82 , obj.: 14161.362 Vol.: 0.390, ch.: 0.068

it.: 83 , obj.: 14159.208 Vol.: 0.390, ch.: 0.063

it.: 84 , obj.: 14157.785 Vol.: 0.390, ch.: 0.065

it.: 85 , obj.: 14155.651 Vol.: 0.390, ch.: 0.074

it.: 86 , obj.: 14152.511 Vol.: 0.390, ch.: 0.084

it.: 87 , obj.: 14150.430 Vol.: 0.390, ch.: 0.095

it.: 88 , obj.: 14148.782 Vol.: 0.390, ch.: 0.106

it.: 89 , obj.: 14147.134 Vol.: 0.390, ch.: 0.118

it.: 90 , obj.: 14143.849 Vol.: 0.390, ch.: 0.053

it.: 91 , obj.: 14141.954 Vol.: 0.390, ch.: 0.050

it.: 92 , obj.: 14141.047 Vol.: 0.390, ch.: 0.048

it.: 93 , obj.: 14138.238 Vol.: 0.390, ch.: 0.047

it.: 94 , obj.: 14135.354 Vol.: 0.390, ch.: 0.050

it.: 95 , obj.: 14133.907 Vol.: 0.390, ch.: 0.058

it.: 96 , obj.: 14131.475 Vol.: 0.390, ch.: 0.068

it.: 97 , obj.: 14127.980 Vol.: 0.390, ch.: 0.078

it.: 98 , obj.: 14125.158 Vol.: 0.390, ch.: 0.089

it.: 99 , obj.: 14122.487 Vol.: 0.390, ch.: 0.100

it.: 100 , obj.: 14119.269 Vol.: 0.390, ch.: 0.111

Press any key...

OBERVATION/ CONCLUSION :

Hence the optimization for the problem formulation is done and the result are obtained.