```
import gurobipy as gp
from gurobipy import GRB
import re

def readinfiles(mpsfilename, constrfilename):
    """
    This function takes the mps file to extract the information about the problem
    and uses the file containing information about hard constraints to slice the
constraints
    :param mpsfilename: the name of the mps file
    :param constrfilename: the file about the hard constraints
    :return: A1, b1: the hard constraints and rhs values
             A2, b2: the soft constraints and rhs values
             objcoeff: the coefficient for variables from the objective function
             lpr.objVal: objective value from Linear relaxed program
             -1*mipexact.ObjVal: objective value by solving exactly
             directopt: objective value by solving the mps file exactly
    """
    m = gp.read(mpsfilename)
    m.Params.LogToConsole = 0
    m.optimize()
    directopt = m.objVal

    # convert the constraint information from mps file into matrix type
    num_rows = m.numconstrs
    num_col = m.numvars
    coeffmatrix = [None] * num_rows

    for k in range(num_rows):
        constrslist = [None] * num_col
        for i in range(num_col):
            constrslist[i] = m.getCoeff(m.getConstrs()[k], m.getVars()[i])
        coeffmatrix[k] = constrslist

    rhside = m.rhs  # right hand side values of the constraints
    objcoeff = m.obj  # coefficient in the objective function

    # print(objcoeff)
    # print(coeffmatrix[0])
    # print(rhside)

    # Now solve the Mixed-integer Program exactly, and check if the output is the
same as directly running the mps file
    mipexact = gp.Model()
    mipexact.Params.LogToConsole = 0
    x = []
    for i in range(len(objcoeff)):
        x.append(mipexact.addVar(vtype=GRB.BINARY))

    for j in range(len(rhside)):
        mipexact.addConstr(sum(coeffmatrix[j][i] * x[i] for i in range(len(x))) <=
rhside[j])

    mipexact.setObjective(-1*sum(objcoeff[m] * x[m] for m in range(len(x))),
GRB.MAXIMIZE)
    mipexact.optimize()

    print("Successfully draw information from the mps file!")
    print("For mixed-integer program, optimal solution is: ", mipexact.X)
    # Next solve the relaxed version
    lpr = gp.Model()
    lpr.Params.LogToConsole = 0
    x = []
    for i in range(len(objcoeff)):
        x.append(lpr.addVar(vtype=GRB.CONTINUOUS))

    for j in range(len(rhside)):
        lpr.addConstr(sum(coeffmatrix[j][i] * x[i] for i in range(len(x))) <=
rhside[j])

    lpr.setObjective(sum(objcoeff[m] * x[m] for m in range(len(x))))
    lpr.optimize()

    print("For LP relaxation, optimal solution is: ", lpr.X)

    # Suppose we have a file that indicates which constraints are hard
    hardconstridx = []
    with open(constrfilename, 'r') as file:
        # Loop over each line in the file
        for line in file:
            # Use regular expressions to find the numbers in the line
            numbers = re.findall(r'\d+', line)
            hardconstridx.append(int(numbers[0]))

    # separate the constraints into hard ones and soft ones according to the
information above
    A1 = coeffmatrix[hardconstridx[0]:hardconstridx[-1] + 1]
    b1 = rhside[hardconstridx[0]:hardconstridx[-1] + 1]
    A2 = coeffmatrix[hardconstridx[-1] + 1:]
    b2 = rhside[hardconstridx[-1] + 1:]

    return A1, b1, A2, b2, objcoeff, lpr.objVal , mipexact.ObjVal, directopt

def LagrangianDualSolver(initlmbd,hardcoeff,hardrhs,softcoeff, softrhs, objcoef,
lpopt, mipopt1):
    '''
    This is the Lagrangian Dual Solver.
    :param initlmbd: the initial λ we choose
    :param hardcoeff: coefficient matrix for hard constraints
    :param hardrhs:  rhs value for hard constraints
    :param softcoeff: coefficient matrix for soft constraints
    :param softrhs: rhs value for soft constraints
    :param objcoef: c for objective function
    :param lpopt: linear relax program objective value
    :param mipopt1: mixed-integer program objective value
    :return:
    '''
    Aa1 = hardcoeff
    bb1 = hardrhs
    Aa2 = softcoeff
    bb2 = softrhs
    objc = objcoef
    zlp = lpopt
    zi1 = mipopt1
    # Next solve the Lagrangian dual problem
    ldp = gp.Model()
    ldp.Params.LogToConsole = 0
    x = []
    for i in range(len(objc)):
        x.append(ldp.addVar(vtype=GRB.BINARY))

    # Initialize the s subgradient
    s_grad = [ldp.addVar() for i in range(len(bb1))]

    # Express hard constraints as b1-A1*x
    for j in range(len(bb1)):
        ldp.addConstr(bb1[j] - sum(Aa1[j][i] * x[i] for i in range(len(x))) ==
s_grad[j])

    # Add the soft constraints
    for j in range(len(bb2)):
        ldp.addConstr(sum(Aa2[j][i] * x[i] for i in range(len(x))) <= bb2[j])

    # Initialize the multiplier
    lmbd = [initlmbd] * len(bb1)

    tol = 1e-6
    iter = 0
    while True:
        # Set the new objective function each time, here s_gradient is just b1-A1*x
        ldp.setObjective((-1)*(sum(objc[m] * x[m] for m in range(len(x)))) +
                         sum(lmbd[n] * s_grad[n] for n in range(len(lmbd))),
GRB.MAXIMIZE)

        ldp.optimize()

        print("Iteration ", iter+1)
        print("Obj value is: ", -1*ldp.objVal)

        satisfy = True
        # Check if satisfy the stopping criteria
        for ll, s in zip(lmbd, s_grad):
            if abs(ll) > tol and abs(s.x) > tol:
                satisfy = False
                break
        if satisfy:
            break
        else:
            a = 1/(iter+1)  # naive step size
            for i in range(len(lmbd)):
                lmbd[i] = max(lmbd[i] - a * s_grad[i].x, 0.0)  # In this case, all
constraints signs are <=, λ is therefore positive

            iter += 1

    print("After extracting information about constraints from mps file and solving
MIP again exactly, obj value is: ",-1*zi1)
    print("Final zLD is ", -1*ldp.objVal)
    print("zLP is ", zlp)
    print("Duality gap is ", (-1*zi1) -zlp)
    print("Sum up, we can observe that zI >= zLD >= zLP. Because this is a
minimization problem,\nwe can conclude that "
      "Lagrangian relaxation provides a bound that is at least as strong as the LP
relaxation." )


# Extract information from the mps file
```

```
[coef1, rhs1, coef2, rhs2, cfobj, lpz, iz1, iz2] = readinfiles('lseu.mps',
'hardconstraints.txt')
# Run the solver
LagrangianDualSolver(4, coef1, rhs1, coef2, rhs2, cfobj,lpz,iz1)
```