optimize.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
MODULE: optimize contains functions for unconstrained minimization
Version: November 29, 2023
Author: Tom Asaki

Functions:
    minimize
    LineSearch
    zoom
    TrustRegionStep
    posroot
    NMstep
    GAstep
    SetDefaults
    ShowResult
"""


#################################################################
def minimize(alg):

    '''
    optimize is the main minimization routine for unconstrained
    smooth-objective problems.

    INPUTS:
        alg     dictionary containing all algorithmic parameters:
            'obj' objective function (handle)
            'x0'  initial guess
            'params'   parameters to pass to the objective function
            'progress' [1]  positive integer.  Progress will be displayed to
                            the terminal after every (progress) iterations.
            'method' [BFGS] string indicating the optimization method to use:
                            'GradientDescent', 'ConjugateGradient', 'BFGS',
                            'LBFGS', 'TrustRegion','NelderMead','Genetic'
            'linesearch' ['Armijo'] string indicating the type of line search
                                    to perform: 'Armijo', 'StrongWolfe'.
            'maxiter' [inf] maximum number of decision variable updates
            'ngtol' [1E-8] stop tolerance on gradient norm
            'dftol' [1E-8] stop tolerance on change in objective
            'dxtol' [1E-8] stop tolerance on change in decision variable norm
            'Lambda' [1] line search initial step length
            'Lambdamax' [100] maximum line search step length
            'c1' [0.001] Armijo sufficient decrease condition parameter
                         ( 0 < c1 < 1/2 )
            'c2' [0.9] Curvature condition parameter
                       ( 0 < c1 < c2 < 1) or
                       ( 0 < c1 < c2 < 1/2) of ConjugateGradient
            'm' [7] number of L-BFGS iterations to save
            'CGreset' [0.2] orthogonality reset tolerance for CG
            'delta' [1]  initial trust region size
            'deltamax' [100] maximum trust region size
            'deltatol' [1E-8] stop tolerance on trust region size
            'eta' [0.01,0.25,0.75] trust region parameters
                                   ( sufficient decrease  shrink , expand )
                                   ( 0 <= eta1 < eta2 < eta3 < 1)
            'maxcond' [1000] maximum condition number on approximate model
                             hessian for trust region method
            'NMpar' [1,2 ,0.5,0.5] refection, expansion, contraction, shrink
                                  parameters for Nelder-Mead
            'NMdiam' [1E-8] Stop tolerance on NM Simplex diameter
            'GApar' [ ceil(sqrt(n)) , 0.1 , 1 ] Genetic Algorithm parameters
                       survival count , mutation rate , mutation size
            'GAstop' [30*n]  stop GA after this many stagnant populations

    OUTPUTS:
        res     a dictionary containing all initial input values and
                additional results of the optimization procedure
            'pr'        copy of the input dictionary with added default
                        values and possibly other algorithmic necessary
                        changes.
            'x'         (n by iter) array whose columns are the decision
                        variable vectors at each iteration
            'f'         (1 by iter) array whose entries are the corresponding
                        objective function values/
            'g'         (n by iter) array whose columns are the gradient
                        vectors at each iteration
            'feval'     total number of function evaluations
            'geval'     total number of gradient evaluations
            'msg'       output message - reason for algorithm termination
    '''

    ##### INITIALIZATIONS ##################################################

    import numpy as np
    from numpy.linalg import norm
    from datetime import datetime

    # Set Default Values for algorithmic parameters as needed
    alg=SetDefaults(alg)
    obj=alg['obj']
    x0=alg['x0']
    params=alg['params']

    # set initial values for output result dictionary res.  Different actions
    # are taken for DFO and nonDFO methods
    match alg['method']:
        case 'NelderMead':
            DFO=True
            Y=x0
            n,m=Y.shape
            fnm=[0]*m
            for k in range(m):
                fnm[k]=obj(Y[:,[k,]],params,1)
            fidx=np.argsort(fnm)
            fnm=[fnm[fidx[e]] for e in range(m)]
            f=fnm[0]
            Y=Y[:,fidx]
            g=np.zeros((n,1))
            ops={'reflect': 0  ,
                 'expand':  0  ,
                 'inside': 0  ,
                 'outside':  0  ,
                 'shrink':  0  }
            res={'x':       Y[:,[0,]]         ,
                 'f':       f                 ,
                 'g':       np.zeros((n,1))   ,
                 'alg':     alg               ,
                 'msg':     ''                ,
                 'feval':   m+1               ,
                 'geval':   0                 ,
                 'operation':  ops            }
        case 'Genetic':
            DFO=True
            Y=x0
            n,m=Y.shape
            fnm=[0]*m
            for k in range(m):
                fnm[k]=obj(Y[:,[k,]],params,1)
            fidx=np.argsort(fnm)
            fnm=[fnm[fidx[e]] for e in range(m)]
            f=fnm[0]
            Y=Y[:,fidx]
            g=np.zeros((n,1))
            res={'x':       Y[:,[0,]]         ,
                 'f':       f                 ,
                 'g':       np.zeros((n,1))   ,
                 'alg':     alg               ,
                 'msg':     ''                ,
                 'feval':   m+1               ,
                 'geval':   0                 ,
                 'generation': 0              }
        case _:
            DFO=False
            f,g=obj(x0,params,2)
            f=np.array([f])
            res={'x':       x0  ,
                 'f':       f   ,
                 'g':       g   ,
                 'alg':     alg ,
                 'msg':     ''  ,
                 'feval':   1   ,
                 'geval':   1   }

    # Initialize iterations
    iter=0
    n=len(x0)

    # Display Initialization
    if alg['progress']:
        print('');
        print('    date      time     iter        f           |g|          |ap|   |df| ');
        print('________________________________________________________________________________');

        dt=datetime.now()
        dtstr=dt.strftime("%Y-%m-%d  %H:%M:%S")
        print('%s  %5d  %+7.4e  %+6.4e ' % (dtstr,iter,f,norm(g)))

    ##### Main Routine #####################################################

    while len(res['msg'])==0:

        match alg['method']:

            case 'GradientDescent':
                # use negative gradient direction as descent direction d
                d=-res['g'][:,[iter,]]
                # compute new best point in direction d
                xnew,flag,nf,ng=LineSearch(res['x'][:,[iter,]],
                                           res['f'][iter],
                                           res['g'][:,[iter,]],
                                           d,
                                           alg,
                                           obj,
                                           params)
                res['feval']+=nf
                res['geval']+=ng

            case 'ConjugateGradient':
                g1=res['g'][:,[iter,]]
                if iter==0:
                    d=-g1
                else:
                    g0=res['g'][:,[iter-1,]]
                    rfactor=np.abs(g1.T.dot(g0))/g1.T.dot(g1)
                    if rfactor>=alg['CGreset']:
                        d=-g1
                    else:
                        beta=(g1.T.dot(g1-g0))/g0.T.dot(g0)
                        beta=np.maximum(beta,0)
                        d=-g1+beta*d

                xnew,flag,nf,ng=LineSearch(res['x'][:,[iter,]],
                                           res['f'][iter],
                                           g1,
                                           d,
                                           alg,
                                           obj,
                                           params)
                res['feval']+=nf
                res['geval']+=ng

            case 'BFGS':
                g1=res['g'][:,[iter,]]
                n=len(g1)
                if iter==0:
                    d=-g1
                    scale=np.maximum(abs(f),np.sqrt(alg['dftol']))
                    H=scale*np.eye(n)
                else:
                    s=res['x'][:,[iter,]]-res['x'][:,[iter-1,]]
                    y=g1-res['g'][:,[iter-1,]]
                    if y.T.dot(s)>norm(s)*norm(y)*(1E-4):
                        r=1/(y.T.dot(s))
                        I=np.eye(n)
                        t1=I-r*(s.dot(y.T))
                        t2=I-r*(y.dot(s.T))
                        t3=r*s.dot(s.T)
                        H=t1.dot(H).dot(t2)+t3
                    d=-H.dot(g1)
                alg['alpha']=1
                xnew,flag,nf,ng=LineSearch(res['x'][:,[iter,]],
                                           res['f'][iter],
                                           g1,
                                           d,
                                           alg,
                                           obj,
                                           params)
                res['feval']+=nf
                res['geval']+=ng

            case 'LBFGS':
                g1=res['g'][:,[iter,]]
                if iter==0:
                    scale=np.maximum(abs(f),np.sqrt(alg['dftol']))
                    d=-scale*g1
                    s=np.zeros((n,0))
                    y=np.zeros((n,0))
                    alph=np.zeros((alg['m'],1))
                else:
                    s=np.hstack((s,res['x'][:,[-1,]]-res['x'][:,[-2,]]))
                    y=np.hstack((y,res['g'][:,[-1,]]-res['g'][:,[-2,]]))
                    if iter>alg['m']:
                        s=np.delete(s,0,1)
                        y=np.delete(y,0,1)
                    d=-res['g'][:,[-1,]]
                    tidx=s.shape[1]-1
                    for i in range(tidx,-1,-1):
                        alph[i]=s[:,i].dot(d)/s[:,i].dot(y[:,i])
                        d-=alph[i]*y[:,[i,]]
                    d=(s[:,tidx].dot(y[:,tidx])/y[:,tidx].dot(y[:,tidx]))*d
                    for i in range(tidx+1):
                        bet=y[:,i].dot(d)/s[:,i].dot(y[:,i])
                        d+=(alph[i]-bet)*s[:,[i,]]

                alg['alpha']=1
                xnew,flag,nf,ng=LineSearch(res['x'][:,[iter,]],
                                          res['f'][iter],
                                          g1,
```

```python
                                          d,
                                          alg,
                                          obj,
                                          params)
                res['feval']+=nf
                res['geval']+=ng


            case 'TrustRegion':

                # update model hession using quasi-Newton ideas
                if iter==0:
                    scale=np.maximum(abs(f),np.sqrt(alg['dftol']))
                    B=scale*np.eye(n)
                else:
                    s=res['x'][:,[-1,]]-res['x'][:,[-2,]]
                    y=res['g'][:,[-1,]]-res['g'][:,[-2,]]
                    w=y-B.dot(s)
                    B=B+(w.dot(w.T))/(w.T.dot(s))

                # call the trust region step algorithm
                xx=res['x'][:,[-1,]]
                ff=res['f'][-1]
                gg=res['g'][:,[-1,]]
                p=alg['params']
                xnew,flag,nf,delta=TrustRegionStep(xx,ff,gg,B,alg,p)
                res['feval']+=nf

                # save the trust region size for the next iteration
                alg['delta']=delta

            case 'NelderMead':

                # Call a Nelder-Mead step
                Y,fnm,nf,flag,ops=NMstep(Y,fnm,alg)
                res['feval']+=nf
                xnew=Y[:,[0,]]
                res['operation']['reflect']+=ops[0]
                res['operation']['expand']+=ops[1]
                res['operation']['inside']+=ops[2]
                res['operation']['outside']+=ops[3]
                res['operation']['shrink']+=ops[4]

            case 'Genetic':

                # Call a Genetic Algorithm step
                Y,fnm,nf,flag,generation=GAstep(Y,fnm,alg)
                res['feval']+=nf
                xnew=Y[:,[0,]]
                res['generation']+=generation

        # Update iteration counter
        iter+=1

        # Update x,f,g
        res['x']=np.append(res['x'],xnew,1)
        if not DFO:
            ff,gg=obj(xnew,params,2)
            res['geval']+=1
            res['f']=np.append(res['f'],ff)
            res['g']=np.append(res['g'],gg,1)
        else:
            res['f']=np.append(res['f'],fnm[0])
            ff=fnm[0]
            gg=0

        # check termination criteria
        if iter>alg['maxiter']:
            res['msg']='Maximum number of iterations reached.'
        if norm(res['g'][:,-1])<alg['ngtol']:
            res['msg']='Minimum gradient norm reached.'
        if (iter>0 and norm(res['x'][:,iter-1]-res['x'][:,iter])<alg['dxtol']):
            res['msg']='Minimum step size reached.'
        if (iter>0 and np.abs(res['f'][iter-1]-res['f'][iter])<alg['dftol']):
            res['msg']='Minimum change in objective reached.'
        if flag:
            match alg['method']:
                case 'TrustRegion':
                    res['msg']='Minimum trust region size reached.'
                case 'NelderMead':
                    res['msg']='Minimum Simplex size reached.'
                case 'Genetic':
                    res['msg']='Number of stagnant GA populations reached.'
                case _:
                    res['msg']='Linesearch failed to find an acceptable iterate.'

        # Show Progress
        if alg['progress']:
            if len(res['msg'])>0 or not np.mod(iter,alg['progress']):
                dt=datetime.now()
                dtstr=dt.strftime("%Y-%m-%d  %H:%M:%S")
                gg=norm(gg)
                ap=norm(res['x'][:,-2]-res['x'][:,-1])
                df=res['f'][-2]-res['f'][-1]
                print('%s  %5d  %+7.4e  %+6.4e  %+6.4e  %+6.4e' % (dtstr,iter,ff,gg,ap,df))

    # Finalize progress
    if alg['progress'] and len(res['msg'])>0:
        print('──────────────────────────────────────────────────────────────────────────────');
        print('')
        dt=datetime.now()
        dtstr=dt.strftime("%Y-%m-%d  %H:%M:%S")
        print('%s  %s' % (dtstr,res['msg']))

    return res

#################################################################
```

```
def SetDefaults(alg):
    import numpy as np
    n=len(alg['x0'])
    alg.setdefault( 'method',     'BFGS'  )
    alg.setdefault( 'linesearch', 'StrongWolfe')
    alg.setdefault( 'maxiter',    np.inf  )
    alg.setdefault( 'ngtol',      1E-8    )
    alg.setdefault( 'dftol',      1E-8    )
    alg.setdefault( 'dxtol',      1E-8    )
    alg.setdefault( 'Lambda',     1       )
    alg.setdefault( 'Lambdamax',  100     )
    alg.setdefault( 'c1',         0.0001  )
    alg.setdefault( 'c2',         0.9     )
    alg.setdefault( 'm',          7       )
    alg.setdefault( 'CGreset',    0.2     )
    alg.setdefault( 'delta',      1       )
    alg.setdefault( 'deltamax',   100     )
    alg.setdefault( 'deltatol',   1E-8    )
    alg.setdefault( 'eta',        [0.01,0.25,0.75]  )
    alg.setdefault( 'maxcond',    1000    )
    alg.setdefault( 'NMpar',      [1,2,0.5,0.5]    )
    alg.setdefault( 'GApar',      [ int(np.ceil(np.sqrt(n))),0.1,1])
    alg.setdefault( 'GAstop',     30*n   )
    alg.setdefault( 'progress',   1 )

    if alg['method']=='ConjugateGradient' and alg['c2']>=0.5:
        alg['c2']=0.4
    if alg['c1']>=alg['c2']:
        alg['c1']=0.0001
    alg['m']=np.minimum(alg['m'],alg['x0'].size)
    if alg['method']=='NelderMead':
        alg['ngtol']=0
    if alg['method']=='Genetic':
        alg['dxtol']=0
        alg['ngtol']=0
        alg['dftol']=0

    return alg

#################################################################
def LineSearch(x,f,g,d,alg,obj,p):

    import numpy as np
    from numpy.linalg import norm

    nf=0
    ng=0

    match alg['linesearch']:

        case 'Armijo':
            goflag=True
            flag=False
            L=alg['Lambda']
            c1=alg['c1']
            dx=alg['dxtol']
            d0=d.T.dot(g).item()
            while goflag:
                xnew=x+L*d
                fnew=obj(xnew,p,1)
                nf+=1
                if fnew>f+c1*L*d0:
                    L/=2
                    if norm(L*d)<dx:
                        goflag=False
                        flag=True
                else:
                    goflag=False
            return xnew,flag,nf,ng

        case 'StrongWolfe':
            goflag=True
            flag=False
            k=0
            L=[]
            L.append(alg['Lambda'])
            c1=alg['c1']
            c2=alg['c2']
            dxtol=alg['dxtol']
            d0=d.T.dot(g).item()
            F=[]
            while goflag:
                F.append(obj(x+L[k]*d,p,1))
                nf+=1
                if F[k]>f+c1*L[k]*d0 or (k>0 and F[k]>=F[k-1]):
                    if k==0:
                        lambdastar,nnf,nng=zoom(0,L[k],x,f,d,d0,f,p,c1,c2,dxtol,obj)
                        nf+=nnf
                        ng+=nng
                    else:
                        lambdastar,nnf,nng=zoom(L[k-1],L[k],x,f,d,d0,F[k-1],p,c1,c2,dxtol,obj)
                        nf+=nnf
                        ng+=nng
                    goflag=False
                if goflag:
                    dummy,g=obj(x+L[k]*d,p,2)
                    nf+=1
                    ng+=1
                    dk=d.T.dot(g).item()
                    if np.abs(dk)<=-c2*d0:
                        lambdastar=L[k]
                        goflag=False
                if goflag and dk>=0:
                    if k==0:
                        lambdastar,nnf,nng=zoom(L[k],0,x,f,d,d0,F[k],p,c1,c2,dxtol,obj)
                        nf+=nnf
                        ng+=nng
                    else:
                        lambdastar,nnf,nng=zoom(L[k],L[k-1],x,f,d,d0,F[k],p,c1,c2,dxtol,obj)
                        nf+=nnf
                        ng+=nng
                    goflag=False
                if goflag:
                    k+=1
                    L.append(2*L[k-1])
                    if L[k]>alg['Lambdamax']:
                        flag=True
                        goflag=False
                        lambdastar=0
            xnew=x+lambdastar*d
            return xnew,flag,nf,ng

        case _:
            return

################################################################
def zoom(L,H,x,f,d,d0,fL,p,c1,c2,dxtol,obj):
    import numpy as np
    from numpy.linalg import norm
    nnf=0
    nng=0
    goflag=True
    while goflag:
        M=(L+H)/2
        fM=obj(x+M*d,p,1)
        nnf+=1
        if fM>f+c1*M*d0 or fM>=fL:
            H=M
        else:
            dummy,gM=obj(x+M*d,p,2)
            nnf+=1
            nng+=1
            dk=d.T.dot(gM).item()
            if np.abs(dk)<=-c2*d0:
                lambdastar=M
                goflag=False
            if goflag:
                if dk*(H-L)>=0:
                    H=L
                L=M
        if M*norm(d)<2*dxtol:
            lambdastar=M
            goflag=False
    return lambdastar,nnf,nng

################################################################
def TrustRegionStep(x,f,g,B,alg,p):

    import numpy as np
    from numpy.linalg import norm

    flag=False
    obj=alg['obj']
    delta=alg['delta']
    rho=-1;
    nf=0;
    e1,e2,e3=alg['eta']
    n=x.size

    # loop until an acceptable point is found or the trust region
    # becomes too small
    while rho<=e1:

        # compute the Steihaug-Toint Step
        z=np.zeros((n,1))
        r=g.copy()
        d=-g.copy()
        ng=norm(g)
        eterm=np.minimum(0.5,np.sqrt(ng))*ng
        StopCondition=False
        inneriter=0

        while not StopCondition:
            t=d.T.dot(B.dot(d))
            if t<=0:
                tau=posroot(z,d,delta)
                step=z+tau*d
                StopCondition=True
            else:
                alpha=(r.T.dot(r))/t
                z+=alpha*d
                if norm(z)>=delta:
                    tau=posroot(z,d,delta)
                    step=z+tau*d
                    StopCondition=True
                else:
                    rold=r.copy()
                    r+=alpha*(B.dot(d))
                    if norm(r)<eterm:
                        step=z
                        StopCondition=True
                    beta=(r.T.dot(r))/(rold.T.dot(rold))
                    d=-r+beta*d
            inneriter+=1
            if inneriter==n:
                step=z
                StopCondition=True

        # evaluate rho (reliability parameter)
        fnew=obj(x+step,p,1)
        nf+=1
        modelchange=-g.T.dot(step)-0.5*step.T.dot(B.dot(step))
        rho=(f-fnew)/modelchange

        # updates: shrink or grow delta, keep an improved point
        if rho<e2:
            delta/=4
        else:
            if rho>e3 and norm(step)>0.999*delta:
                delta=np.minimum(2*delta,alg['deltamax'])
        if rho>e1:
            xnew=x+step

        # if trust region gets too small, then stop with no result
        if delta<alg['deltatol']:
            flag=True
            rho=1
            xnew=x+step

    # if, for any reason fnew>=f, then do not update x
    if fnew>=f:
        xnew=x

    return xnew, flag, nf, delta

#################################################################
def posroot(z,d,delta):
    import numpy as np
    a=z.T.dot(d)
    b=d.T.dot(d)
    tau=-(a/b)+np.sqrt((a/b)**2+(delta**2-z.T.dot(z))/b)
    return tau

#################################################################
def NMstep(Y,fnm,alg):
    import numpy as np

    # set parameters
    n=len(Y)
    NMre,NMex,NMco,NMsh=alg['NMpar']
    obj=alg['obj']
    p=alg['params']
    ops=[0,0,0,0,0]

    # initialize loop variables.  nf is the number of function evaluations,
    # fbe is thee best objective value known at the time of call, flag is
    # set to True if the stagnant generation limit is reached.
    nf=0
    fbe=fnm[0]
    goflag=True
    flag=False
    while goflag:

        # assume for now that the shrink step will not occur
        shrinkit=False

        # Compute some geometrically relevant vectors
        Centroid=np.sum(Y[:,0:n],axis=1).reshape((n,1))/n
        Best=Y[:,[0,]]
        Worst=Y[:,[-1,]]
        Line=Centroid-Worst

        # Compute the reflection point
        Reflect=Worst+(NMre+1)*Line
        fre=obj(Reflect,p,1)
        nf+=1

        # If Reflect is the new best, then take the best of Reflect and Expand
        if fre<fbe:

            Expand=Worst+(NMex+1)*Line
            fex=obj(Expand,p,1)
            nf+=1
            if fex<=fre:
                y=Expand
                f=fex
                ops[1]+=1
            else:
                y=Reflect
                f=fre
                ops[0]+=1

        # If Reflect improves on the second worst point, then keep it
        elif fre<fnm[-2]:

            y=Reflect
            f=fre
            ops[0]+=1

        # If Reflect is the new second worst then try outside contract
        elif fre<fnm[-1]:

            Outside=Worst+(NMco+1)*Line
            foc=obj(Outside,p,1)
            nf+=1

            if foc<fre:

                y=Outside
                f=foc
                ops[3]+=1

            else:

                y=Reflect
                f=fre
                ops[0]+=1

        # If Reflect is the new worst then try inside contract
        else:

            Inside=Worst+(1-NMco)*Line
            fic=obj(Inside,p,1)
            nf+=1

            if fic<fnm[-1]:

                y=Inside
                f=fic
                ops[2]+=1

            else:

                shrinkit=True

        # Update the simplex by either arranging the new point into the
        # simplex data or by performing a simplex shrink
        if shrinkit:

            Y=Y+NMsh*(Best-Y)
            for k in range(1,n+1):
                fnm[k]=obj(Y[:,[k,]],p,1)
            nf+=n
            fidx=np.argsort(fnm)
            fnm=fnm[fidx]
            Y=Y[:,fidx]
            y=Y[:,[0,]]
            f=fnm[0]
            ops[4]+=1

        else:

            idx=next(i for i in range(n+1) if fnm[i] > f)
            fnm=np.insert(fnm,idx,f)
            fnm=np.delete(fnm,[-1])
            Y=np.concatenate((Y[:,0:idx],y,Y[:,idx:n]),axis=1)

        # Check stopping criteria: an improved best point
        if fnm[0]<fbe:
            goflag=False
        if np.linalg.norm(y-Best)<alg['dxtol']:
            goflag=False
            flag=True

    return Y,fnm,nf,flag,ops

#################################################################
def GAstep(Y,fnm,alg):

    import numpy as np

    #Set genetic algorithm parameters
    n,m=Y.shape
    GAsu,GAmr,GAmw=alg['GApar']
    obj=alg['obj']
    p=alg['params']

    # Initialize loop parameters.  nf is the number of objective evaluations,
    # fbe is the best objective value on calling this function, generation
    # is the generation counter, flag is ture if stagnation is reached.
    nf=0
    generation=0;
    fbe=fnm[0]
    goflag=True
    flag=False

    while goflag:

        # keep the direct survivors
        Ynew=np.zeros(Y.shape)
        Ynew[:,0:GAsu]=Y[:,0:GAsu]
        fnew=np.concatenate((fnm[0:GAsu],[np.inf]*(m-GAsu)))

        # Compute fitness scores
        a=1;
        F=fnm[-1]-fnm+a
        CumProb=np.cumsum(F)
        CumProb=CumProb/CumProb[-1]

        # Add to the new generation by building offspring
        for k in range(GAsu,m):

            # Choose distinct parents from the current population
            R=np.random.rand(1)
            P1=next(i for i in range(m) if R<CumProb[i])
            P2=P1
            while P1==P2:
                R=np.random.rand(1)
                P2=next(i for i in range(m) if R<CumProb[i])

            # Build offspring as convex combimation of parents with
            # weights determined by parent fitnesses
            theta=F[P1]/(F[P1]+F[P2])
            y=theta*Y[:,[P1]]+(1-theta)*Y[:,[P2]]

            # Apply gene mutation (by coordinate values
            for j in range(n):
                if np.random.rand(1)<GAmr:
                    y[j]+=GAmw*np.random.randn(1)

            # add offspring to the new generation and determine the
            # corresponding objective value
            f=obj(y,p,1)
            nf+=1
            idx=next(i for i in range(m) if f<fnew[i])
            fnew=np.insert(fnew,idx,f)
            fnew=np.delete(fnew,[-1])
            Ynew=np.concatenate((Ynew[:,0:idx],y,Ynew[:,idx:m-1]),axis=1)

        # check stopping criteria.  Either a new best individual is found
        # or the population is stagnant
        Y=Ynew
        fnm=fnew
        generation+=1
        if fnm[0]<fbe:
            goflag=False
        elif generation==alg['GAstop']:
            goflag=False
            flag=True


    return Y,fnm,nf,flag,generation

#################################################################
def ShowResults(res):
    import numpy as np
    n,iter=res['x'].shape
    print('')
    print('--------------------------------')
    print('')
    print('Optimal Objective = %f' % (res['f'][iter-1]))
    print('')
    print('Nonzero Optimal Variables:')
    for k in range(n):
        if np.abs(res['x'][k,iter-1])>1E-8:
            print(' x(%2d) = %f' % (k+1,res['x'][k,iter-1]))
    print('')
    print('--------------------------------')
    print('')
    return

```