optimize.py

```
#!/usr/bin/env python3
 2
   # -*- coding: utf-8 -*-
 3
  MODULE: optimize contains functions for unconstrained minimization
 4
   Version: November 29, 2023
  Author: Tom Asaki
 6
 7
 8
  Functions:
9
       minimize
10
       LineSearch
11
       zoom
12
       TrustRegionStep
13
       posroot
14
       NMstep
       GAstep
15
       SetDefaults
16
17
       ShowResult
   0.000
18
19
20
21
   22
   def minimize(alg):
23
24
25
       optimize is the main minimization routine for unconstrained
26
       smooth-objective problems.
27
28
       INPUTS:
           alg
                   dictionary containing all algorithmic parameters:
29
               'obj' objective function (handle)
30
                'x0' initial guess
31
                          parameters to pass to the objective function
32
                'params'
                'progress' [1] positive integer. Progress will be displayed to
33
                               the terminal after every (progress) iterations.
34
35
                'method' [BFGS] string indicating the optimization method to use:
                               'GradientDescent', 'ConjugateGradient', 'BFGS',
36
37
                               'LBFGS', 'TrustRegion', 'NelderMead', 'Genetic'
                'linesearch' ['Armijo'] string indicating the type of line search
38
39
                                       to perform: 'Armijo', 'StrongWolfe'.
                'maxiter' [inf] maximum number of decision variable updates
40
                'ngtol' [1E-8] stop tolerance on gradient norm
41
                'dftol' [1E-8] stop tolerance on change in objective
42
43
                'dxtol' [1E-8] stop tolerance on change in decision variable norm
                'Lambda' [1] line search initial step length
44
                'Lambdamax' [100] maximum line search step length
45
46
                'c1' [0.001] Armijo sufficient decrease condition parameter
                            (0 < c1 < 1/2)
47
                'c2' [0.9] Curvature condition parameter
48
49
                          (0 < c1 < c2 < 1) or
50
                          ( 0 < c1 < c2 < 1/2) of ConjugateGradient
                'm' [7] number of L-BFGS iterations to save
51
52
               'CGreset' [0.2] orthogonality reset tolerance for CG
```

```
53
                 'delta' [1] initial trust region size
                 'deltamax' [100] maximum trust region size
54
                 'deltatol' [1E-8] stop tolerance on trust region size
55
                 'eta' [0.01,0.25,0.75] trust region parameters
56
                                        ( sufficient decrease shrink , expand )
57
58
                                        ( 0 \le \text{eta1} \le \text{eta2} \le \text{eta3} \le 1 )
                 'maxcond' [1000] maximum condition number on approximate model
59
                                  hessian for trust region method
60
                 'NMpar' [1,2 ,0.5,0.5] refection, expansion, contraction, shrink
61
                                       parameters for Nelder-Mead
62
                 'NMdiam' [1E-8] Stop tolerance on NM Simplex diameter
63
                 'GApar' [ ceil(sqrt(n)) , 0.1 , 1 ] Genetic Algorithm parameters
64
65
                            survival count , mutation rate , mutation size
66
                 'GAstop' [30*n] stop GA after this many stagnant populations
67
68
        OUTPUTS:
69
                    a dictionary containing all initial input values and
            res
70
                    additional results of the optimization procedure
                 'pr'
                             copy of the input dictionary with added default
71
72
                             values and possibly other algorithmic necessary
73
                             changes.
                             (n by iter) array whose columns are the decision
74
75
                             variable vectors at each iteration
                 'f'
                             (1 by iter) array whose entries are the corresponding
76
77
                            objective function values/
                             (n by iter) array whose columns are the gradient
78
                 'g'
79
                            vectors at each iteration
                 'feval'
                            total number of function evaluations
80
                 'geval'
                            total number of gradient evaluations
81
82
                 'msg'
                            output message - reason for algorithm termination
         1.1.1
83
84
85
        86
87
        import numpy as np
        from numpy.linalg import norm
88
        from datetime import datetime
89
90
91
        # Set Default Values for algorithmic parameters as needed
        alg=SetDefaults(alg)
92
93
        obj=alg['obj']
        x0=alg['x0']
94
        params=alg['params']
95
96
        # set initial values for output result dictionary res. Different actions
97
98
        # are taken for DFO and nonDFO methods
        match alg['method']:
99
100
            case 'NelderMead':
                DF0=True
101
102
                Y=x0
103
                n,m=Y.shape
                fnm=[0]*m
104
105
                for k in range(m):
                    fnm[k]=obj(Y[:,[k,]],params,1)
106
                fidx=np.argsort(fnm)
107
```

```
108
                  fnm=[fnm[fidx[e]] for e in range(m)]
109
                  f=fnm[0]
110
                  Y=Y[:,fidx]
111
                  g=np.zeros((n,1))
                  ops={'reflect': 0
112
                        'expand': 0
113
114
                        'inside': 0
115
                        'outside': 0
                        'shrink':
116
                                   0 }
                  res={'x':
117
                                   Y[:,[0,]]
                        'f':
118
                                   f
                        'g':
                                   np.zeros((n,1))
119
                        'alg':
120
                                   alg
121
                        'msg':
                        'feval':
122
                                   m+1
123
                        'geval':
                                   0
124
                        'operation': ops
125
             case 'Genetic':
126
                  DF0=True
127
                  Y=x0
128
                  n,m=Y.shape
                  fnm=[0]*m
129
                  for k in range(m):
130
                      fnm[k]=obj(Y[:,[k,]],params,1)
131
                  fidx=np.argsort(fnm)
132
                  fnm=[fnm[fidx[e]] for e in range(m)]
133
                  f=fnm[0]
134
                  Y=Y[:,fidx]
135
                  g=np.zeros((n,1))
136
                  res={'x':
137
                                   Y[:,[0,]]
                        'f':
138
                                   f
139
                        'g':
                                   np.zeros((n,1))
140
                        'alg':
                                   alg
141
                        'msg':
142
                        'feval':
                                   m+1
                        'geval':
143
                                   0
                                                       }
144
                        'generation': 0
145
             case _:
146
                  DF0=False
147
                  f,g=obj(x0,params,2)
                  f=np.array([f])
148
                  res={'x':
149
                                   х0
                        'f':
150
                                   f
151
                        'g':
                                   g
                                   alg ,
152
                        'alg':
153
                        'msg':
154
                        'feval':
                                   1
155
                        'geval':
                                        }
                                   1
156
157
         # Initialize iterations
158
         iter=0
         n=len(x0)
159
160
161
         # Display Initialization
         if alg['progress']:
162
```

```
print('');
163
164
            print('
                       date
                                 time
                                                     f
                                                                  |g|
                                                                             |ap|
                                          iter
     |df| ');
165
            print('
    );
166
            dt=datetime.now()
            dtstr=dt.strftime("%Y-%m-%d %H:%M:%S")
167
            print('%s %5d %+7.4e %+6.4e ' % (dtstr,iter,f,norm(g)))
168
169
        170
171
        while len(res['msg'])==0:
172
173
            match alg['method']:
174
175
176
                case 'GradientDescent':
177
                    # use negative gradient direction as descent direction d
                    d=-res['g'][:,[iter,]]
178
179
                    # compute new best point in direction d
180
                    xnew,flag,nf,ng=LineSearch(res['x'][:,[iter,]],
                                               res['f'][iter],
181
182
                                               res['g'][:,[iter,]],
183
                                               d,
184
                                               alg,
185
                                               obj,
186
                                               params)
                    res['feval']+=nf
187
188
                    res['geval']+=ng
189
190
                case 'ConjugateGradient':
                    g1=res['g'][:,[iter,]]
191
                    if iter==0:
192
193
                        d=-g1
                    else:
194
195
                        g0=res['g'][:,[iter-1,]]
                        rfactor=np.abs(g1.T.dot(g0))/g1.T.dot(g1)
196
197
                        if rfactor>=alg['CGreset']:
198
                            d=-g1
199
                        else:
                            beta=(g1.T.dot(g1-g0))/g0.T.dot(g0)
200
201
                            beta=np.maximum(beta,0)
202
                            d=-g1+beta*d
203
                    xnew,flag,nf,ng=LineSearch(res['x'][:,[iter,]],
204
                                               res['f'][iter],
205
206
                                               g1,
                                               d,
207
208
                                               alg,
209
                                               obj,
210
                                               params)
                    res['feval']+=nf
211
212
                    res['geval']+=ng
213
214
                case 'BFGS':
215
                    g1=res['g'][:,[iter,]]
```

```
216
                     n=len(g1)
217
                     if iter==0:
218
                          d=-g1
                          scale=np.maximum(abs(f),np.sqrt(alg['dftol']))
219
                          H=scale*np.eye(n)
220
                     else:
221
                          s=res['x'][:,[iter,]]-res['x'][:,[iter-1,]]
222
                          y=g1-res['g'][:,[iter-1,]]
223
                          if y.T.dot(s)>norm(s)*norm(y)*(1E-4):
224
225
                              r=1/(y.T.dot(s))
226
                              I=np.eye(n)
                              t1=I-r*(s.dot(y.T))
227
                              t2=I-r*(y.dot(s.T))
228
229
                              t3=r*s.dot(s.T)
                              H=t1.dot(H).dot(t2)+t3
230
231
                          d=-H.dot(g1)
232
                     alg['alpha']=1
233
                     xnew,flag,nf,ng=LineSearch(res['x'][:,[iter,]],
                                                  res['f'][iter],
234
235
                                                  g1,
236
                                                  d,
237
                                                  alg,
238
                                                  obj,
                                                  params)
239
                     res['feval']+=nf
240
                     res['geval']+=ng
241
242
                 case 'LBFGS':
243
                     g1=res['g'][:,[iter,]]
244
245
                     if iter==0:
246
                          scale=np.maximum(abs(f),np.sqrt(alg['dftol']))
247
                          d=-scale*g1
248
                          s=np.zeros((n,0))
                          y=np.zeros((n,0))
249
250
                          alph=np.zeros((alg['m'],1))
251
                     else:
252
                          s=np.hstack((s,res['x'][:,[-1,]]-res['x'][:,[-2,]]))
                          y=np.hstack((y,res['g'][:,[-1,]]-res['g'][:,[-2,]]))
253
                          if iter>alg['m']:
254
255
                              s=np.delete(s,0,1)
                              y=np.delete(y,0,1)
256
                          d=-res['g'][:,[-1,]]
257
258
                          tidx=s.shape[1]-1
259
                          for i in range(tidx,-1,-1):
                              alph[i]=s[:,i].dot(d)/s[:,i].dot(y[:,i])
260
261
                              d-=alph[i]*y[:,[i,]]
262
                          d=(s[:,tidx].dot(y[:,tidx])/y[:,tidx].dot(y[:,tidx]))*d
                          for i in range(tidx+1):
263
                              bet=y[:,i].dot(d)/s[:,i].dot(y[:,i])
264
265
                              d+=(alph[i]-bet)*s[:,[i,]]
266
                     alg['alpha']=1
267
                     xnew,flag,nf,ng=LineSearch(res['x'][:,[iter,]],
268
269
                                                 res['f'][iter],
270
                                                 g1,
```

```
271
                                                  d,
                                                 alg,
272
273
                                                  obj,
274
                                                  params)
                      res['feval']+=nf
275
276
                      res['geval']+=ng
277
278
279
                 case 'TrustRegion':
280
281
                      # update model hession using quasi-Newton ideas
                      if iter==0:
282
                          scale=np.maximum(abs(f),np.sqrt(alg['dftol']))
283
284
                          B=scale*np.eye(n)
285
                      else:
286
                          s=res['x'][:,[-1,]]-res['x'][:,[-2,]]
287
                          y=res['g'][:,[-1,]]-res['g'][:,[-2,]]
288
                          w=y-B.dot(s)
                          B=B+(w.dot(w.T))/(w.T.dot(s))
289
290
291
                      # call the trust region step algorithm
                      xx=res['x'][:,[-1,]]
292
                      ff=res['f'][-1]
293
294
                      gg=res['g'][:,[-1,]]
295
                      p=alg['params']
296
                      xnew,flag,nf,delta=TrustRegionStep(xx,ff,gg,B,alg,p)
297
                      res['feval']+=nf
298
299
                      # save the trust region size for the next iteration
                      alg['delta']=delta
300
301
302
                 case 'NelderMead':
303
304
                      # Call a Nelder-Mead step
305
                      Y, fnm, nf, flag, ops=NMstep(Y, fnm, alg)
306
                      res['feval']+=nf
307
                      xnew=Y[:,[0,]]
                      res['operation']['reflect']+=ops[0]
308
                      res['operation']['expand']+=ops[1]
309
                      res['operation']['inside']+=ops[2]
310
                      res['operation']['outside']+=ops[3]
311
                      res['operation']['shrink']+=ops[4]
312
313
                 case 'Genetic':
314
315
316
                      # Call a Genetic Algorithm step
                      Y, fnm, nf, flag, generation=GAstep(Y, fnm, alg)
317
                      res['feval']+=nf
318
                      xnew=Y[:,[0,]]
319
320
                      res['generation']+=generation
321
322
             # Update iteration counter
323
             iter+=1
324
325
             # Update x,f,g
```

```
res['x']=np.append(res['x'],xnew,1)
326
            if not DFO:
327
328
                ff,gg=obj(xnew,params,2)
                res['geval']+=1
329
                res['f']=np.append(res['f'],ff)
330
                res['g']=np.append(res['g'],gg,1)
331
332
            else:
                res['f']=np.append(res['f'],fnm[0])
333
                ff=fnm[0]
334
335
                gg=0
336
337
            # check termination criteria
            if iter>alg['maxiter']:
338
339
                res['msg']='Maximum number of iterations reached.'
            if norm(res['g'][:,-1])<alg['ngtol']:</pre>
340
                res['msg']='Minimum gradient norm reached.'
341
            if (iter>0 and norm(res['x'][:,iter-1]-res['x'][:,iter])<alg['dxtol']):</pre>
342
343
                res['msg']='Minimum step size reached.'
            if (iter>0 and np.abs(res['f'][iter-1]-res['f'][iter])<alg['dftol']):</pre>
344
345
                res['msg']='Minimum change in objective reached.'
346
            if flag:
                match alg['method']:
347
                    case 'TrustRegion':
348
                        res['msg']='Minimum trust region size reached.'
349
                    case 'NelderMead':
350
                        res['msg']='Minimum Simplex size reached.'
351
352
                    case 'Genetic':
                        res['msg']='Number of stagnant GA populations reached.'
353
354
                    case _:
355
                        res['msg']='Linesearch failed to find an acceptable
    iterate.'
356
357
            # Show Progress
            if alg['progress']:
358
                if len(res['msg'])>0 or not np.mod(iter,alg['progress']):
359
                    dt=datetime.now()
360
                    dtstr=dt.strftime("%Y-%m-%d %H:%M:%S")
361
                    gg=norm(gg)
362
363
                    ap=norm(res['x'][:,-2]-res['x'][:,-1])
364
                    df=res['f'][-2]-res['f'][-1]
365
                    print('%s %5d %+7.4e %+6.4e %+6.4e %+6.4e' % (dtstr,iter,
    ff,gg,ap,df))
366
367
        # Finalize progress
        if alg['progress'] and len(res['msg'])>0:
368
            print('
369
    );
            print('')
370
            dt=datetime.now()
371
            dtstr=dt.strftime("%Y-%m-%d %H:%M:%S")
372
            print('%s %s' % (dtstr,res['msg']))
373
374
375
        return res
376
377
```

```
378
    def SetDefaults(alg):
379
        import numpy as np
380
        n=len(alg['x0'])
                                       'BFGS'
381
        alg.setdefault( 'method',
        alg.setdefault( 'linesearch', 'StrongWolfe')
382
        alg.setdefault( 'maxiter',
383
                                       np.inf
                                               )
        alg.setdefault( 'ngtol',
384
                                       1E-8
                                               )
385
        alg.setdefault( 'dftol',
                                       1E-8
                                               )
386
        alg.setdefault( 'dxtol',
                                       1E-8
                                               )
        alg.setdefault( 'Lambda',
387
                                       1
                                               )
                                               )
        alg.setdefault( 'Lambdamax',
                                       100
388
        alg.setdefault( 'c1',
                                               )
389
                                       0.0001
                                       0.9
390
        alg.setdefault( 'c2',
        alg.setdefault( 'm',
                                       7
391
                                               )
        alg.setdefault( 'CGreset',
                                       0.2
392
393
        alg.setdefault( 'delta',
                                       1
                                               )
        alg.setdefault( 'deltamax',
394
                                       100
                                               )
395
        alg.setdefault( 'deltatol',
                                       1E-8
                                               )
        alg.setdefault( 'eta',
                                       [0.01,0.25,0.75]
396
397
        alg.setdefault( 'maxcond',
                                       1000
        alg.setdefault( 'NMpar',
398
                                       [1,2,0.5,0.5]
                                      [ int(np.ceil(np.sqrt(n))),0.1,1])
399
        alg.setdefault( 'GApar',
        alg.setdefault( 'GAstop',
400
                                       30*n
        alg.setdefault( 'progress',
                                       1)
401
402
        if alg['method']=='ConjugateGradient' and alg['c2']>=0.5:
403
             alg['c2']=0.4
404
        if alg['c1']>=alg['c2']:
405
             alg['c1']=0.0001
406
        alg['m']=np.minimum(alg['m'],alg['x0'].size)
407
408
        if alg['method'] == 'NelderMead':
409
            alg['ngtol']=0
410
        if alg['method'] == 'Genetic':
411
            alg['dxtol']=0
412
            alg['ngtol']=0
413
            alg['dftol']=0
414
415
        return alg
416
417
    418
    def LineSearch(x,f,g,d,alg,obj,p):
419
420
        import numpy as np
421
        from numpy.linalg import norm
422
423
        nf=0
        ng=0
424
425
        match alg['linesearch']:
426
427
428
             case 'Armijo':
429
                goflag=True
                flag=False
430
431
                L=alg['Lambda']
                c1=alg['c1']
432
```

```
433
                  dx=alg['dxtol']
434
                  d0=d.T.dot(g).item()
435
                  while goflag:
436
                       xnew=x+L*d
437
                       fnew=obj(xnew,p,1)
438
                       nf+=1
                       if fnew>f+c1*L*d0:
439
440
                           L/=2
                           if norm(L*d)<dx:</pre>
441
442
                                goflag=False
443
                                flag=True
444
                       else:
445
                           goflag=False
446
                  return xnew, flag, nf, ng
447
448
              case 'StrongWolfe':
449
                  goflag=True
450
                  flag=False
451
                  k=0
452
                  L=[]
453
                  L.append(alg['Lambda'])
                  c1=alg['c1']
454
                  c2=alg['c2']
455
                  dxtol=alg['dxtol']
456
                  d0=d.T.dot(g).item()
457
                  F=[]
458
459
                  while goflag:
                       F.append(obj(x+L[k]*d,p,1))
460
461
                       nf+=1
462
                       if F[k]>f+c1*L[k]*d0 or (k>0 and F[k]>=F[k-1]):
463
                           if k==0:
464
                                lambdastar,nnf,nng=zoom(0,L[k],x,f,d,d0,f,p,c1,c2,dxtol,
     obj)
465
                               nf+=nnf
466
                               ng+=nng
467
                           else:
468
                                lambdastar, nnf, nng=zoom(L[k-1], L[k], x, f, d, d0, F[k-1], p,
     c1,c2,dxtol,obj)
469
                               nf+=nnf
470
                                ng+=nng
471
                           goflag=False
472
                       if goflag:
473
                           dummy, g = obj(x + L[k] * d, p, 2)
474
                           nf+=1
475
                           ng+=1
476
                           dk=d.T.dot(g).item()
477
                           if np.abs(dk) < = -c2*d0:
478
                               lambdastar=L[k]
479
                                goflag=False
480
                       if goflag and dk>=0:
                           if k==0:
481
482
                                lambdastar,nnf,nng=zoom(L[k],0,x,f,d,d0,F[k],p,c1,c2,
     dxtol,obj)
483
                               nf+=nnf
484
                                ng+=nng
485
                           else:
```

```
486
                            lambdastar, nnf, nng=zoom(L[k], L[k-1], x, f, d, d0, F[k], p, c1,
    c2,dxtol,obj)
487
                            nf+=nnf
488
                            ng+=nng
489
                        goflag=False
                    if goflag:
490
491
                        k+=1
                        L.append(2*L[k-1])
492
493
                        if L[k]>alg['Lambdamax']:
494
                            flag=True
                            goflag=False
495
496
                            lambdastar=0
497
                xnew=x+lambdastar*d
498
                return xnew, flag, nf, ng
499
500
            case _:
501
                return
502
503
    504
    def zoom(L,H,x,f,d,d0,fL,p,c1,c2,dxtol,obj):
505
        import numpy as np
506
        from numpy.linalg import norm
        nnf=0
507
508
        nng=0
509
        goflag=True
510
        while goflag:
            M = (L + H)/2
511
512
            fM=obj(x+M*d,p,1)
513
            nnf+=1
514
            if fM>f+c1*M*d0 or fM>=fL:
515
                H = M
516
            else:
517
                dummy, gM = obj(x+M*d,p,2)
                nnf+=1
518
519
                nng+=1
                dk=d.T.dot(gM).item()
520
521
                if np.abs(dk) < = -c2*d0:
                    lambdastar=M
522
523
                    goflag=False
524
                if goflag:
525
                    if dk*(H-L)>=0:
526
                        H=L
527
                    L = M
            if M*norm(d)<2*dxtol:</pre>
528
529
                lambdastar=M
530
                goflag=False
531
        return lambdastar, nnf, nng
532
533
    534
    def TrustRegionStep(x,f,g,B,alg,p):
535
536
        import numpy as np
537
        from numpy.linalg import norm
538
539
        flag=False
```

```
540
         obj=alg['obj']
541
         delta=alg['delta']
542
         rho=-1;
543
         nf=0;
544
         e1,e2,e3=alg['eta']
545
         n=x.size
546
547
         # loop until an acceptable point is found or the trust region
548
         # becomes too small
549
         while rho<=e1:</pre>
550
551
             # compute the Steihaug-Toint Step
             z=np.zeros((n,1))
552
553
             r=g.copy()
             d=-g.copy()
554
555
             ng=norm(g)
556
             eterm=np.minimum(0.5,np.sqrt(ng))*ng
557
             StopCondition=False
558
             inneriter=0
559
560
             while not StopCondition:
561
                  t=d.T.dot(B.dot(d))
                  if t<=0:
562
563
                      tau=posroot(z,d,delta)
                      step=z+tau*d
564
565
                      StopCondition=True
566
                  else:
                      alpha=(r.T.dot(r))/t
567
                      z+=alpha*d
568
569
                      if norm(z)>=delta:
570
                           tau=posroot(z,d,delta)
571
                           step=z+tau*d
572
                          StopCondition=True
573
                      else:
574
                          rold=r.copy()
                           r+=alpha*(B.dot(d))
575
576
                           if norm(r)<eterm:</pre>
577
                               step=z
578
                               StopCondition=True
579
                          beta=(r.T.dot(r))/(rold.T.dot(rold))
                          d=-r+beta*d
580
                  inneriter+=1
581
582
                  if inneriter==n:
583
                      step=z
584
                      StopCondition=True
585
586
             # evaluate rho (reliability parameter)
587
             fnew=obj(x+step,p,1)
             nf+=1
588
589
             modelchange=-g.T.dot(step)-0.5*step.T.dot(B.dot(step))
590
             rho=(f-fnew)/modelchange
591
592
             # updates: shrink or grow delta, keep an improved point
593
             if rho<e2:</pre>
                  delta∕=4
594
```

```
595
            else:
596
                if rho>e3 and norm(step)>0.999*delta:
597
                   delta=np.minimum(2*delta,alg['deltamax'])
598
            if rho>e1:
599
                xnew=x+step
600
601
            # if trust region gets too small, then stop with no result
            if delta<alg['deltatol']:</pre>
602
603
                flag=True
604
                rho=1
605
                xnew=x+step
606
        # if, for any reason fnew>=f, then do not update x
607
        if fnew>=f:
608
            xnew=x
609
610
611
        return xnew, flag, nf, delta
612
    613
614
    def posroot(z,d,delta):
615
        import numpy as np
616
        a=z.T.dot(d)
        b=d.T.dot(d)
617
        tau=-(a/b)+np.sqrt((a/b)**2+(delta**2-z.T.dot(z))/b)
618
619
        return tau
620
    621
622
    def NMstep(Y,fnm,alg):
623
        import numpy as np
624
625
        # set parameters
626
        n=len(Y)
627
        NMre,NMex,NMco,NMsh=alg['NMpar']
        obj=alg['obj']
628
629
        p=alg['params']
630
        ops=[0,0,0,0,0]
631
        # initialize loop variables. nf is the number of function evaluations,
632
        # fbe is thee best objective value known at the time of call, flag is
633
        # set to True if the stagnant generation limit is reached.
634
        nf=0
635
        fbe=fnm[0]
636
637
        goflag=True
        flag=False
638
639
        while goflag:
640
            # assume for now that the shrink step will not occur
641
642
            shrinkit=False
643
644
            # Compute some geometrically relevant vectors
645
            Centroid=np.sum(Y[:,0:n],axis=1).reshape((n,1))/n
            Best=Y[:,[0,]]
646
            Worst=Y[:,[-1,]]
647
648
            Line=Centroid-Worst
649
```

```
650
              # Compute the reflection point
              Reflect=Worst+(NMre+1)*Line
651
652
              fre=obj(Reflect,p,1)
653
              nf+=1
654
              # If Reflect is the new best, then take the best of Reflect and Expand
655
656
              if fre<fbe:</pre>
657
658
                  Expand=Worst+(NMex+1)*Line
659
                  fex=obj(Expand,p,1)
                  nf+=1
660
                  if fex<=fre:</pre>
661
662
                       y=Expand
                       f=fex
663
                       ops[1]+=1
664
665
                  else:
                       y=Reflect
666
667
                       f=fre
                       ops[0]+=1
668
669
670
              # If Reflect improves on the second worst point, then keep it
              elif fre<fnm[-2]:</pre>
671
672
                  y=Reflect
673
                  f=fre
674
                  ops[0]+=1
675
676
              # If Reflect is the new second worst then try outside contract
677
              elif fre<fnm[-1]:</pre>
678
679
680
                  Outside=Worst+(NMco+1)*Line
681
                  foc=obj(Outside,p,1)
682
                  nf+=1
683
684
                  if foc<fre:</pre>
685
686
                       y=Outside
                       f=foc
687
                       ops[3]+=1
688
689
                  else:
690
691
692
                       y=Reflect
693
                       f=fre
                       ops[0]+=1
694
695
696
              # If Reflect is the new worst then try inside contract
697
              else:
698
                  Inside=Worst+(1-NMco)*Line
699
700
                  fic=obj(Inside,p,1)
                  nf+=1
701
702
                  if fic<fnm[-1]:</pre>
703
704
```

```
705
                     y=Inside
706
                     f=fic
707
                     ops[2]+=1
708
709
                 else:
710
711
                     shrinkit=True
712
713
            # Update the simplex by either arranging the new point into the
714
            # simplex data or by performing a simplex shrink
            if shrinkit:
715
716
                 Y=Y+NMsh*(Best-Y)
717
                 for k in range(1,n+1):
718
                     fnm[k]=obj(Y[:,[k,]],p,1)
719
720
                 nf+=n
721
                 fidx=np.argsort(fnm)
722
                 fnm=fnm[fidx]
                 Y=Y[:,fidx]
723
724
                 y=Y[:,[0,]]
725
                 f=fnm[0]
                 ops[4]+=1
726
727
728
            else:
729
                 idx=next(i for i in range(n+1) if fnm[i] > f)
730
                 fnm=np.insert(fnm,idx,f)
731
                 fnm=np.delete(fnm,[-1])
732
                 Y=np.concatenate((Y[:,0:idx],y,Y[:,idx:n]),axis=1)
733
734
735
            # Check stopping criteria: an improved best point
736
            if fnm[0]<fbe:</pre>
737
                 goflag=False
738
            if np.linalg.norm(y-Best)<alg['dxtol']:</pre>
739
                 goflag=False
740
                 flag=True
741
742
        return Y, fnm, nf, flag, ops
743
744
    745
    def GAstep(Y,fnm,alg):
746
747
        import numpy as np
748
749
        #Set genetic algorithm parameters
750
        n, m=Y. shape
751
        GAsu,GAmr,GAmw=alg['GApar']
752
        obj=alg['obj']
753
        p=alg['params']
754
755
        # Initialize loop parameters. nf is the number of objective evaluations,
756
        # fbe is the best objective value on calling this function, generation
757
        # is the generation counter, flag is ture if stagnation is reached.
758
        nf=0
759
        generation=0;
```

```
760
         fbe=fnm[0]
761
         goflag=True
762
         flag=False
763
764
         while goflag:
765
766
             # keep the direct survivors
             Ynew=np.zeros(Y.shape)
767
768
             Ynew[:,0:GAsu]=Y[:,0:GAsu]
769
             fnew=np.concatenate((fnm[0:GAsu],[np.inf]*(m-GAsu)))
770
             # Compute fitness scores
771
772
             a=1;
773
             F=fnm[-1]-fnm+a
             CumProb=np.cumsum(F)
774
775
             CumProb=CumProb/CumProb[-1]
776
777
             # Add to the new generation by building offspring
778
             for k in range(GAsu,m):
779
780
                 # Choose distinct parents from the current population
781
                 R=np.random.rand(1)
782
                 P1=next(i for i in range(m) if R<CumProb[i])
783
                 P2=P1
784
                 while P1==P2:
785
                     R=np.random.rand(1)
786
                     P2=next(i for i in range(m) if R<CumProb[i])
787
788
                 # Build offspring as convex combination of parents with
                 # weights determined by parent fitnesses
789
790
                 theta=F[P1]/(F[P1]+F[P2])
791
                 y=theta*Y[:,[P1]]+(1-theta)*Y[:,[P2]]
792
793
                 # Apply gene mutation (by coordinate values
794
                 for j in range(n):
795
                     if np.random.rand(1)<GAmr:</pre>
796
                          y[j]+=GAmw*np.random.randn(1)
797
798
                 # add offspring to the new generation and determine the
799
                 # corresponding objective value
                 f=obj(y,p,1)
800
                 nf+=1
801
802
                 idx=next(i for i in range(m) if f<fnew[i])
                 fnew=np.insert(fnew,idx,f)
803
804
                 fnew=np.delete(fnew,[-1])
805
                 Ynew=np.concatenate((Ynew[:,0:idx],y,Ynew[:,idx:m-1]),axis=1)
806
             # check stopping criteria. Either a new best individual is found
807
808
             # or the population is stagnant
809
             Y=Ynew
810
             fnm=fnew
811
             generation+=1
             if fnm[0]<fbe:</pre>
812
813
                 goflag=False
             elif generation==alg['GAstop']:
814
```

```
815
               goflag=False
               flag=True
816
817
818
819
       return Y, fnm, nf, flag, generation
820
    821
    def ShowResults(res):
822
823
       import numpy as np
       n,iter=res['x'].shape
824
       print('')
825
826
       print('—
827
       print('')
       print('Optimal Objective = %f' % (res['f'][iter-1]))
828
       print('')
829
       print('Nonzero Optimal Variables:')
830
       for k in range(n):
831
           if np.abs(res['x'][k,iter-1])>1E-8:
832
               print(' x(%2d) = %f' % (k+1,res['x'][k,iter-1]))
833
       print('')
834
       print('---
835
       print('')
836
       return
837
838
```