

optimize.py

```

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

```

```

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

```

```

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

```

```
163         print('');
164         print('      date      time      iter      f      |g|      |lap|
|df| ');
165         print('
');
166         dt=datetime.now()
167         dtstr=dt.strftime("%Y-%m-%d %H:%M:%S")
168         print('%s %5d %+.4e %+.4e ' % (dtstr,iter,f,norm(g)))
169
170     ##### Main Routine #####
171
172     while len(res['msg'])==0:
173
174         match alg['method']:
175
176             case 'GradientDescent':
177                 # use negative gradient direction as descent direction d
178                 d=-res['g'][:,[iter,]]
179                 # compute new best point in direction d
180                 xnew,flag,nf,ng=LineSearch(res['x'][:,[iter,]],
181                                         res['f'][iter],
182                                         res['g'][:,[iter,]],
183                                         d,
184                                         alg,
185                                         obj,
186                                         params)
187
188                 res['feval']+=nf
189                 res['geval']+=ng
190
191             case 'ConjugateGradient':
192                 g1=res['g'][:,[iter,]]
193                 if iter==0:
194                     d=-g1
195                 else:
196                     g0=res['g'][:,[iter-1,]]
197                     rfactor=np.abs(g1.T.dot(g0))/g1.T.dot(g1)
198                     if rfactor>=alg['CGreset']:
199                         d=-g1
200                     else:
201                         beta=(g1.T.dot(g1-g0))/g0.T.dot(g0)
202                         beta=np.maximum(beta,0)
203                         d=-g1+beta*d
204
205                 xnew,flag,nf,ng=LineSearch(res['x'][:,[iter,]],
206                                         res['f'][iter],
207                                         g1,
208                                         d,
209                                         alg,
210                                         obj,
211                                         params)
212
213                 res['feval']+=nf
214                 res['geval']+=ng
215
216             case 'BFGS':
217                 g1=res['g'][:,[iter,]]
```

```

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

```

```
271                                     d,
272                                     alg,
273                                     obj,
274                                     params)
275     res['feval']+=nf
276     res['geval']+=ng
277
278
279     case 'TrustRegion':
280
281         # update model hessian using quasi-Newton ideas
282         if iter==0:
283             scale=np.maximum(abs(f),np.sqrt(alg['dftol']))
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)
289             B=B+(w.dot(w.T))/(w.T.dot(s))
290
291         # call the trust region step algorithm
292         xx=res['x'][:,[-1,]]
293         ff=res['f'][-1]
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
300         alg['delta']=delta
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,]]
308         res['operation']['reflect']+=ops[0]
309         res['operation']['expand']+=ops[1]
310         res['operation']['inside']+=ops[2]
311         res['operation']['outside']+=ops[3]
312         res['operation']['shrink']+=ops[4]
313
314     case 'Genetic':
315
316         # Call a Genetic Algorithm step
317         Y,fnm,nf,flag,generation=GAstep(Y,fnm,alg)
318         res['feval']+=nf
319         xnew=Y[:,[0,]]
320         res['generation']+=generation
321
322     # Update iteration counter
323     iter+=1
324
325     # Update x,f,g
```

```

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

```

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



```

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
439             if fnew>f+c1*L*d0:
440                 L/=2
441                 if norm(L*d)<dx:
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'])
454         c1=alg['c1']
455         c2=alg['c2']
456         dxtol=alg['dxtol']
457         d0=d.T.dot(g).item()
458         F=[]
459         while goflag:
460             F.append(obj(x+L[k]*d,p,1))
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:
481                 if k==0:
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
490         if goflag:
491             k+=1
492             L.append(2*L[k-1])
493             if L[k]>alg['Lambdamax']:
494                 flag=True
495                 goflag=False
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
507     nnf=0
508     nng=0
509     goflag=True
510     while goflag:
511         M=(L+H)/2
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)
518             nnf+=1
519             nng+=1
520             dk=d.T.dot(gM).item()
521             if np.abs(dk)<=-c2*d0:
522                 lambdastar=M
523                 goflag=False
524             if goflag:
525                 if dk*(H-L)>=0:
526                     H=L
527                 L=M
528             if M*norm(d)<2*dxtol:
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:
550
551     # compute the Steihaug-Toint Step
552     z=np.zeros((n,1))
553     r=g.copy()
554     d=-g.copy()
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))
562         if t<=0:
563             tau=posroot(z,d,delta)
564             step=z+tau*d
565             StopCondition=True
566         else:
567             alpha=(r.T.dot(r))/t
568             z+=alpha*d
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()
575                 r+=alpha*(B.dot(d))
576                 if norm(r)<eterm:
577                     step=z
578                     StopCondition=True
579                 beta=(r.T.dot(r))/(rold.T.dot(rold))
580                 d=-r+beta*d
581             inneriter+=1
582             if inneriter==n:
583                 step=z
584                 StopCondition=True
585
586     # evaluate rho (reliability parameter)
587     fnew=obj(x+step,p,1)
588     nf+=1
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:
594         delta/=4
```

```
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
602             if delta<alg['deltatol']:
603                 flag=True
604                 rho=1
605                 xnew=x+step
606
607             # if, for any reason fnew>=f, then do not update x
608             if fnew>=f:
609                 xnew=x
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)
617     b=d.T.dot(d)
618     tau=-(a/b)+np.sqrt((a/b)**2+(delta**2-z.T.dot(z))/b)
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']
628     obj=alg['obj']
629     p=alg['params']
630     ops=[0,0,0,0,0]
631
632     # initialize loop variables.  nf is the number of function evaluations,
633     # fbe is the best objective value known at the time of call, flag is
634     # set to True if the stagnant generation limit is reached.
635     nf=0
636     fbe=fnm[0]
637     goflag=True
638     flag=False
639     while goflag:
640
641         # assume for now that the shrink step will not occur
642         shrinkit=False
643
644         # Compute some geometrically relevant vectors
645         Centroid=np.sum(Y[:,0:n],axis=1).reshape((n,1))/n
646         Best=Y[:,[0,]]
647         Worst=Y[:,[-1,]]
648         Line=Centroid-Worst
649
```

```
650     # Compute the reflection point
651     Reflect=Worst+(NMre+1)*Line
652     fre=obj(Reflect,p,1)
653     nf+=1
654
655     # If Reflect is the new best, then take the best of Reflect and Expand
656     if fre<fbe:
657
658         Expand=Worst+(NMex+1)*Line
659         fex=obj(Expand,p,1)
660         nf+=1
661         if fex<=fre:
662             y=Expand
663             f=fex
664             ops[1]+=1
665         else:
666             y=Reflect
667             f=fre
668             ops[0]+=1
669
670     # If Reflect improves on the second worst point, then keep it
671     elif fre<fnm[-2]:
672
673         y=Reflect
674         f=fre
675         ops[0]+=1
676
677     # If Reflect is the new second worst then try outside contract
678     elif fre<fnm[-1]:
679
680         Outside=Worst+(NMco+1)*Line
681         foc=obj(Outside,p,1)
682         nf+=1
683
684         if foc<fre:
685
686             y=Outside
687             f=foc
688             ops[3]+=1
689
690         else:
691
692             y=Reflect
693             f=fre
694             ops[0]+=1
695
696     # If Reflect is the new worst then try inside contract
697     else:
698
699         Inside=Worst+(1-NMco)*Line
700         fic=obj(Inside,p,1)
701         nf+=1
702
703         if fic<fnm[-1]:
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
715     if shrinkit:
716
717         Y=Y+NMsh*(Best-Y)
718         for k in range(1,n+1):
719             fnm[k]=obj(Y[:,[k,]],p,1)
720         nf+=n
721         fidx=np.argsort(fnm)
722         fnm=fnm[fidx]
723         Y=Y[:,fidx]
724         y=Y[:,[0,]]
725         f=fnm[0]
726         ops[4]+=1
727
728     else:
729
730         idx=next(i for i in range(n+1) if fnm[i] > f)
731         fnm=np.insert(fnm,idx,f)
732         fnm=np.delete(fnm,[-1])
733         Y=np.concatenate((Y[:,0:idx],y,Y[:,idx:n]),axis=1)
734
735     # Check stopping criteria: an improved best point
736     if fnm[0]<fbe:
737         goflag=False
738     if np.linalg.norm(y-Best)<alg['dxtol']:
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
767     Ynew=np.zeros(Y.shape)
768     Ynew[:,0:GAsu]=Y[:,0:GAsu]
769     fnew=np.concatenate((fnm[0:GAsu],[np.inf]*(m-GAsu)))
770
771     # Compute fitness scores
772     a=1;
773     F=fnm[-1]-fnm+a
774     CumProb=np.cumsum(F)
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
789         # weights determined by parent fitnesses
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:
796                 y[j]+=GAmw*np.random.randn(1)
797
798         # add offspring to the new generation and determine the
799         # corresponding objective value
800         f=obj(y,p,1)
801         nf+=1
802         idx=next(i for i in range(m) if f<fnew[i])
803         fnew=np.insert(fnew,idx,f)
804         fnew=np.delete(fnew,[-1])
805         Ynew=np.concatenate((Ynew[:,0:idx],y,Ynew[:,idx:m-1]),axis=1)
806
807     # check stopping criteria. Either a new best individual is found
808     # or the population is stagnant
809     Y=Ynew
810     fnm=fnew
811     generation+=1
812     if fnm[0]<fbe:
813         goflag=False
814     elif generation==alg['GAstop']:
```

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