**ALGORITHMS AND DESIGN ANALYSIS:**

**COURSE PROJECT**

Shahrukh Zarir

Pranav Yadav

Ebrahim Merchant

3D MAPS OF 2 FUNCTIONS PAGES 2-4

PERFORMANCE PLOTS PAGES 5-22

COMPARISION OF DE AND PSO PAGES (WITH SOLUTION) 23-25

IMPLEMENTATION CODE 26-END

**3D Maps of 2D Functions**

|  |  |
| --- | --- |
| **Benchmark Functions** | **3D Maps** |
| High Conditioned Elliptic Function | **Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 1.55.32 PM.png** |
| Bent Cigar Function | **Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 1.57.41 PM.png** |
| Discus Function | **Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 1.58.49 PM.png** |
| Rosenbrock’s Function | **Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 1.59.54 PM.png** |
| Ackley’s Function | **Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 2.01.07 PM.png** |
| Weierstrass Function | **Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 9.00.00 PM.png** |
| Griewank’s Function | **Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 3.46.35 PM.png** |
| Rastrigin’s Function | **Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 2.24.49 PM.png** |
| Katsuura Function | **Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 5.11.53 PM.png** |

**Performance Plots**

**Particle Swarm Optimization**

|  |  |
| --- | --- |
| **Function (D=10)** | ***Performance Plots*** |
| High Conditioned Elliptic Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 5.33.33 PM.png |
| Bent Cigar Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 5.22.51 PM.png |
| Discus Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 5.22.51 PM.png |
| Rosenbrock’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 5.43.45 PM.png |
| Ackley’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 5.26.52 PM.png |
| Weierstrass Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 9.02.41 PM.png |
| Griewank’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 5.28.40 PM.png |
| Rastrigins Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 5.28.40 PM.png |
| Katsuura Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 5.31.39 PM.png |

|  |  |
| --- | --- |
| **Function (D=30)** | ***Performance Plots*** |
| High Conditioned Elliptic Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.08.21 PM.png |
| Bent Cigar Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.08.44 PM.png |
| Discus Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.09.04 PM.png |
| Rosenbrock’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.09.22 PM.png |
| Ackley’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.12.54 PM.png |
| Weierstrass Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.13.37 PM.png |
| Griewank’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.14.01 PM.png |
| Rastrigins Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.14.22 PM.png |
| Katsuura Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.16.27 PM.png |

|  |  |
| --- | --- |
| **Function (D=50)** | ***Performance Plots*** |
| High Conditioned Elliptic Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.23.21 PM.png |
| Bent Cigar Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.24.58 PM.png |
| Discus Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.25.17 PM.png |
| Rosenbrock’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.25.37 PM.png |
| Ackley’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.25.55 PM.png |
| Weierstrass Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.26.36 PM.png |
| Griewank’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.26.55 PM.png |
| Rastrigins Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.27.14 PM.png |
| Katsuura Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-11-30 at 11.29.15 PM.png |

**Differential Evolution**

|  |  |
| --- | --- |
| **Function (D=10)** | ***Performance Plots*** |
| High Conditioned Elliptic Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.22.16 AM.png |
| Bent Cigar Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.22.39 AM.png |
| Discus Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.23.02 AM.png |
| Rosenbrock’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.23.26 AM.png |
| Ackley’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.23.46 AM.png |
| Weierstrass Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.24.24 AM.png |
| Griewank’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.24.48 AM.png |
| Rastrigins Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.25.26 AM.png |
| Katsuura Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.27.36 AM.png |

|  |  |
| --- | --- |
| **Function (D=30)** | ***Performance Plots*** |
| High Conditioned Elliptic Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.30.54 AM.png |
| Bent Cigar Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.31.13 AM.png |
| Discus Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.31.45 AM.png |
| Rosenbrock’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.32.09 AM.png |
| Ackley’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.32.44 AM.png |
| Weierstrass Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.34.13 AM.png |
| Griewank’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.35.06 AM.png |
| Rastrigins Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.37.47 AM.png |
| Katsuura Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.43.01 AM.png |

|  |  |
| --- | --- |
| **Function (D=50)** | ***Performance Plots*** |
| High Conditioned Elliptic Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.47.00 AM.png |
| Bent Cigar Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.47.37 AM.png |
| Discus Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.47.56 AM.png |
| Rosenbrock’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.48.13 AM.png |
| Ackley’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.48.37 AM.png |
| Weierstrass Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.50.03 AM.png |
| Griewank’s Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.50.20 AM.png |
| Rastrigins Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.50.51 AM.png |
| Katsuura Function | Macintosh HD:Users:shahrukhzarir:Desktop:Screen Shot 2016-12-01 at 12.52.20 AM.png |

**DIMENSION 10**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Function | Algorithm | Mean | Standard Deviation | Best Value | Worst Value | Best Solution | Fitness Value |
| f1 | PSO | 18.47969228 | 12.7126512 | 1.43 | 44.303 | 1.43 | [ 2.22304434e-01 4.74413477e-02 1.17432411e+00 3.33695024e-07] |
| f1 | DE | 669.290181 | 3150.67658 | 2.10E-27 | 157768.4057 | 2.10E-27 | [-2.08317121e-14 -4.63501630e-15 1.63458208e-14 -1.27529295e-17] |
| f2 | PSO | 5423382.19 | 5944353.86 | 56559.12094 | 22578993.37 | 56559.12094 | [ 0.00599621 0.23762487 0.00623424 0.00432703] |
| f2 | DE | 15175160.4 | 69796832.6 | 2.65E-23 | 349018816 | 2.65E-23 | [-1.06518823e-15 3.59876446e-16 -5.16523089e-16 2.12233078e-15] |
| f3 | PSO | 21.2017 | 17.8380871 | 0.405175201 | 65.91 | 0.405175201 | [ 0. 0.26473844 0.02582527 0.57829215] |
| f3 | DE | 1444.5194 | 6903.7 | 4.01E-27 | 34552.42472 | 4.01E-27 | [2.52294054e-15 6.09174139e-16 5.64638122e-15 -1.07089356e-14] |
| f4 | PSO | 83.97 | 193.467 | 0.894184664 | 862.74 | 0.894184664 | [ 1.07863975 1.10151977 1.26326255 1.55393791] |
| f4 | DE | 96509.06 | 457698.293 | 1.04E-11 | 2290729.07 | 1.04E-11 | [1.00000004 1.00000009 1.00000014 1.00000031] |
| f5 | PSO | 6.3107 | 3.4764 | 1.296262503 | 14.20200006 | 1.296262503 | [ 0.00496108 0.08205496 1.05273048 0.08188966] |
| f5 | DE | 20.281246 | 0.07701878 | 2.02E+01 | 20.4471145 | 2.02E+01 | 1284.90196702 5070.01318411 -371.96449959 841.02109309] |
| f6 | PSO | 25.198 | 0.7577 | 23.99997711 | 26.465 | 23.99997711 | [ 10. 10. 10. 10.] |
| f6 | DE | 188.532339 | 0.87189877 | 1.87E+02 | 190.081457 | 1.87E+02 | 4382.32399732 920.99204542 2485.99044825 1902.97614334 |
| f7 | PSO | 0.202235112 | 0.1038851 | 0.039976498 | 0.45066128 | 0.039976498 | [ 6.51099447 0. 0.14762712 0. ] |
| f7 | DE | 0.4114126 | 0.06153538 | 3.62E-01 | 1.15174073 | 3.62E-01 | 8.13166511 -7.85363944 9.18551891 8.71288816 -0.59189011 |
| f8 | PSO | 12.17 | 4.1165 | 5.473110427 | 0.45066 | 5.473110427 | [ 1.03281198 0.02378559 1.97135049 0.01324928] |
| f8 | DE | 58.9396773 | 25.500695 | 1.25E+01 | 775.130626 | 1.25E+01 | [-1.06482548 0.04957824 0.02088603 -0.02529081] |
| f9 | PSO | 7.20024 | 4.5691 | 6.030094826 | 7.5722 | 6.030094826 | [ 2.95284483 4.48838962 4.51223345 10. ] |

`

**DIMENSION 30**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Function | Algorithm | Mean | Standard Deviation | Best Value | Worst Value | Best Solution | Fitness Value |
| f1 | PSO | 5837678.72 | 3637722.61 | 98312.7481 | 12026623.4 | 98312.7481 | [ 0.31772693 0.21603886 0. 0.43526619] |
| f1 | DE | 135810.1828 | 154768.9338 | 10648.97034 | 456823.1117 | 10648.97034 | [-1.42011038e+01 4.25800510e-01 1.19916602e+01 -2.50307809e+01] |
| f2 | PSO | 20.3714474 | 14.3931204 | 1.76500932 | 68.6483699 | 1.76500932 | [ 1.04310441e-16 2.03558031e-01 7.94805326e-01 1.04492007e+00] |
| f2 | DE | 16156874077 | 17399416660 | 1008238584 | 55525182781 | 1008238584 | [14.80774449 5.811829 -5.05645087 -3.61893471 4.31295447] |
| f3 | PSO | 176.236933 | 498.419119 | 0.00010053 | 2469.99158 | 0.00010053 | [ 1.0021641 1.0043457 1.00874427 1.01762596] |
| f3 | DE | 72393.50911 | 69585.82076 | 2797.047604 | 197400.9095 | 2797.047604 | [ -9.21135862e-01 5.19681580e+00 -6.64943337e+00 8.44204505e+00] |
| f4 | PSO | 71.7719711 | 135.100898 | 0.86158415 | 429.471678 | 0.86158415 | [ 1.02279138 1.11720308 1.21003737 1.42514212] |
| f4 | DE | 3013231524 | 4879371577 | 20449143.48 | 15339156161 | 20449143.48 | [-4.63435744e+00 3.11026965e+00 1.31371838e+01 1.17209034e+02] |
| f5 | PSO | 6.06123697 | 3.56894338 | 0.61682315 | 16.0974236 | 0.61682315 | [ 0.06508761 0. 0.12516617 0.17698568] |
| f5 | DE | 20.87396498 | 0.076573554 | 20.83645177 | 21.02418589 | 20.83645177 | [400.03941785 -52.20203135 1415.90927105 -693.88005137] |
| f6 | PSO | 46.801637 | 133.526496 | 0.90942429 | 576.6922105 | 0.90942429 | [ 1.11797244 1.20138689 1.50099024 2.27207842] |
| f6 | DE | 1780.11996 | 1.317784953 | 1784.278327 | 1778.443851 | 1784.278327 | [2450.94708641 312.93254199 -148.08908381 -2646.20544362] |
| f7 | PSO | 0.05678821 | 0.03844273 | 0 | 0.14530282 | 0 | [ 0. 0. 0. 0.] |
| f7 | DE | 4.751589058 | 4.686582237 | 1.097259821 | 17.19762807 | 1.097259821 | [-2.72267493 5.57045055 -0.50709748 -0.15482845] |
| f8 | PSO | 15.0956827 | 7.83787028 | 3.00923484 | 30.8482851 | 3.00923484 | [ 0.99279254 0.99005337 0.99872935 0.00893607] |
| f8 | DE | 15118.8739 | 17704.78242 | 966.4411706 | 54766.05776 | 966.4411706 | [0.14668588 -5.4294287 2.2943706 -1.02880543 |
| f9 | PSO |  |  |  |  |  |  |
| f9 | DE |  |  |  |  |  |  |

**DIMENSION 50**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Function | Algorithm | Mean | Standard Deviation | Best Value | Worst Value | Best Solution | Fitness Value |
| f1 | PSO | 18.5670916 | 19.3373931 | 1.073818647 | 70.25371977 | 1.073818647 | [  1.82251821e-02   1.48281517e-01   1.02542222e+00   2.89137286e-06] |
| f1 | DE | 158000.8346 | 128993.9054 | 25370.2486 | 379652.4177 | 25370.2486 | [-3.18271695e+01 -1.56574601e+01 4.05877691e+00 4.55379857e-03] |
| f2 | PSO | 6205202.76 | 5490142.58 | 926274.2659 | 24727825.4 | 926274.2659 | [ 0.05395631  0.06788106  0.17629802  0.94216461] |
| f2 | DE | 5.24E+10 | 38468728747 | 8296073888 | 1.12E+11 | 8296073888 | 3.10430772e+00 -2.33311482e+00 1.80386952e+01 2.35983919e+00] |
| f3 | PSO | 19.5541932 | 14.1275355 | 2.5122615 | 53.1798251 | 2.5122615 | [  1.49120505e-06   4.96844097e-01   9.54291872e-01   1.16392966e+00] |
| f3 | DE | 238660.7039 | 113061.3696 | 63408.52846 | 394158.8473 | 63408.52846 | [9.43993130e+00 3.81158544e+01 6.81999548e+01 -1.22394088e+01] |
| f4 | PSO | 72.7741087 | 265.77655 | 0.008582018 | 1308.966684 | 0.008582018 | [ 0.97953047  0.95923258  0.91972649  0.84530499] |
| f4 | DE | 46122476728 | 11553645276 | 7480063714 | 50490152704 | 7480063714 | [13.05118619 -21.69954911 -42.85447875 91.57622573] |
| f5 | PSO | 5.65870088 | 2.93039622 | 0.632724204 | 13.37488471 | 0.632724204 | [ 0.2153185   0.04765693  0.08093524  0.00563713] |
| f5 | DE | 21.13735604 | 0.026288979 | 21.12842547 | 21.23776124 | 21.12842547 | [-23.41928588 1841.23217462 1653.65183883 -602.9335003 |
| f6 | PSO | 25.3599655 | 0.63489528 | 24.42525105 | 26.44006847 | 24.42525105 | [ 8.          3.01107554  9.          7.99588666] |
| f6 | DE | 4973.341334 | 2.121453127 | 4969.667854 | 4976.603905 | 4969.667854 | [-4071.33056711 -27241.0333532 -18588.99601837 297.93988745] |
| f7 | PSO | 0.07691397 | 0.07289617 | 0.002981627 | 0.249298457 | 0.002981627 | [ 0.07714381  0.00189334  0.00406764  0.00436824] |
| f7 | DE | 13.36214009 | 8.526759933 | 3.442159886 | 28.60755745 | 3.442159886 | [-41.07037073 -12.60350651 -11.14758249 -6.18107315] |
| f8 | PSO | 13.2224188 | 6.01519303 | 3.052765966 | 28.5445664 | 3.052765966 | [0.99479911  0.99420574  0.01846618  0.99595065] |
| f8 | DE | 48898.36312 | 39816.95884 | 8882.455199 | 117561.6039 | 8882.455199 | [-2.04954374e+01 -9.64066723e+00 -1.77278164e+01 1.90220813e+01] |
| f9 | PSO | 5.95467816 | 3.81328131 | 0 | 14.62488529 | 0 | [0,0,0,0] |
| f9 | DE |  |  |  |  |  |  |

Codes

**THIS CODE IS FOR THE GRAPHING PORTION**

**from** mpl\_toolkits.mplot3d **import** Axes3D  
**from** matplotlib **import** cm  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
  
*#functions for project***def** f1(x):  
 *"""High Conditioned Elliptic Function"""* sum = 0.0  
 **for** i **in** range(1, len(x)+1):  
 sum += (10\*\*6)\*\*((i-1)/(len(x)-1)) \* x[i-1]\*\*2  
 **return** sum  
  
**def** f2(x):  
 *"""Bent cigar function"""* sum = 0.0  
 sum += x[0]\*\*2  
 **for** i **in** range(2, len(x)+1):  
 sum += x[i-1]\*\*2  
 sum \*= (10\*\*6)  
 **return** sum  
  
**def** f3(x):  
 *"""discus function"""* sum = 0.0  
 sum += (x[0]\*\*2)\*(10\*\*6)  
 **for** i **in** range(2, len(x)+1):  
 sum += x[i-1]\*\*2  
 **return** sum  
  
**def** f4(x):  
 *"""F8 Rosenbrock's saddle"""* sum = 0.0  
 **for** i **in** range(len(x)-1):  
 sum += 100\*((x[i]\*\*2)-x[i+1])\*\*2+\  
 (1-x[i])\*\*2  
 **return** sum  
  
**def** f5(x):  
 *"""Ackley's Function"""* sum1, sum2 = 0.0, 0.0  
 **for** i **in** range(0, len(x)):  
 sum1 += x[i]\*\*2  
 sum1 = sum1 / float(len(x))  
 **for** i **in** range(0, len(x)):  
 sum2 += np.cos(2\*np.pi\*x[i])  
 sum2 = sum2 / float(len(x))  
  
 *# Calculate first exp* exp1 = -20.0 \* (np.e \*\* (-0.2 \* sum1))  
 exp2 = np.e \*\* sum2  
  
 *# Calculate final result* result = exp1 - exp2 + 20 + np.e  
 **return** result  
  
**def** f6(x):  
 sum1, sum2, sum3 = 0.0, 0.0, 0.0  
 a = 0.5  
 b = 3  
 kmax = 20  
 **for** i **in** range(len(x)):  
 **for** k **in** range(0, kmax):  
 sum2 += (a \*\* k) \* np.cos(2 \* np.pi \* (b \*\* k) \* (x[i] + 0.5))  
 sum3 += (a \*\* k) \* np.cos(2 \* np.pi \* (b \*\* k) \* 0.5)  
 sum1 += sum2 - (len(x) \* sum3)  
 **return** sum1  
  
**def** f7(x):  
 *"""Griewank's function"""* sum = 0  
 **for** i **in** x:  
 sum += i \* i  
 product = 1  
 **for** j **in** xrange(len(x)):  
 product \*= np.cos(x[j] / np.sqrt(j + 1))  
 **return** 1 + sum / 4000 - product  
  
**def** f8(x):  
 *"""Rastrigin's Function"""* sum = 0.0  
 **for** i **in** range(0, len(x)):  
 sum += (x[i]\*\*2 - 10 \* np.cos(2\*np.pi\*x[i]) + 10)  
 **return** sum  
  
**def** f9(x):  
 *"""Katsuura Function"""* product = 1  
 **for** i **in** range(0, len(x)):  
 sum = 0  
 **for** j **in** range(1,33):  
 term = np.power(2,j) \* x[i]  
 sum += np.abs(term - np.round(term))/(np.power(2,j))  
 product \*= np.power(1+((i+1)\*sum),10.0/ np.power(len(x),1.2))  
 **return** (10/len(x) \* len(x) \* product - (10/len(x) \* len(x)))  
  
*#graphs for part 1  
  
#Function 1*X = np.linspace(-100, 100, 100) *# points from 0..10 in the x axis*Y = np.linspace(-100, 100, 100) *# points from 0..10 in the y axis*X, Y = np.meshgrid(X, Y) *# create meshgrid*Z = f1([X, Y]) *# Calculate Z  
  
# Plot the 3D surface for first function from project*fig = plt.figure()  
ax = fig.gca(projection=**'3d'**) *# set the 3d axes*ax.plot\_surface(X, Y, Z,  
 rstride=3,  
 cstride=3,  
 alpha=0.3,  
 cmap=**'hot'**)  
plt.show()  
  
*#Function 2*X = np.linspace(-100, 100, 100) *# points from 0..10 in the x axis*Y = np.linspace(-100, 100, 100) *# points from 0..10 in the y axis*X, Y = np.meshgrid(X, Y) *# create meshgrid*Z = f2([X, Y]) *# Calculate Z  
  
# Plot the 3D surface for first function from project*fig = plt.figure()  
ax = fig.gca(projection=**'3d'**) *# set the 3d axes*ax.plot\_surface(X, Y, Z,  
 rstride=3,  
 cstride=3,  
 alpha=0.3,  
 cmap=**'hot'**)  
plt.show()  
  
*#Function 3*X = np.linspace(-100, 100, 100) *# points from 0..10 in the x axis*Y = np.linspace(-100, 100, 100) *# points from 0..10 in the y axis*X, Y = np.meshgrid(X, Y) *# create meshgrid*Z = f3([X, Y]) *# Calculate Z  
  
# Plot the 3D surface for first function from project*fig = plt.figure()  
ax = fig.gca(projection=**'3d'**) *# set the 3d axes*ax.plot\_surface(X, Y, Z,  
 rstride=3,  
 cstride=3,  
 alpha=0.3,  
 cmap=**'hot'**)  
plt.show()  
  
*#Function 4*X = np.linspace(-100, 100, 100) *# points from 0..10 in the x axis*Y = np.linspace(-100, 100, 100) *# points from 0..10 in the y axis*X, Y = np.meshgrid(X, Y) *# create meshgrid*Z = f4([X, Y]) *# Calculate Z  
  
# Plot the 3D surface for first function from project*fig = plt.figure()  
ax = fig.gca(projection=**'3d'**) *# set the 3d axes*ax.plot\_surface(X, Y, Z,  
 rstride=3,  
 cstride=3,  
 alpha=0.3,  
 cmap=**'hot'**)  
plt.show()  
  
*#Function 5*X = np.linspace(-100, 100, 100) *# points from 0..10 in the x axis*Y = np.linspace(-100, 100, 100) *# points from 0..10 in the y axis*X, Y = np.meshgrid(X, Y) *# create meshgrid*Z = f5([X, Y]) *# Calculate Z  
  
# Plot the 3D surface for first function from project*fig = plt.figure()  
ax = fig.gca(projection=**'3d'**) *# set the 3d axes*ax.plot\_surface(X, Y, Z,  
 rstride=3,  
 cstride=3,  
 alpha=0.3,  
 cmap=**'hot'**)  
plt.show()  
  
*#Function 6*X = np.linspace(-100, 100, 100) *# points from 0..10 in the x axis*Y = np.linspace(-100, 100, 100) *# points from 0..10 in the y axis*X, Y = np.meshgrid(X, Y) *# create meshgrid*Z = f6([X, Y]) *# Calculate Z  
  
# Plot the 3D surface for first function from project*fig = plt.figure()  
ax = fig.gca(projection=**'3d'**) *# set the 3d axes*ax.plot\_surface(X, Y, Z,  
 rstride=3,  
 cstride=3,  
 alpha=0.3,  
 cmap=**'hot'**)  
plt.show()  
  
*#Function 7*X = np.linspace(-100, 100, 100) *# points from 0..10 in the x axis*Y = np.linspace(-100, 100, 100) *# points from 0..10 in the y axis*X, Y = np.meshgrid(X, Y) *# create meshgrid*Z = f7([X, Y]) *# Calculate Z  
  
# Plot the 3D surface for first function from project*fig = plt.figure()  
ax = fig.gca(projection=**'3d'**) *# set the 3d axes*ax.plot\_surface(X, Y, Z,  
 rstride=3,  
 cstride=3,  
 alpha=0.3,  
 cmap=**'hot'**)  
plt.show()  
  
*#Function 8*X = np.linspace(-100, 100, 100) *# points from 0..10 in the x axis*Y = np.linspace(-100, 100, 100) *# points from 0..10 in the y axis*X, Y = np.meshgrid(X, Y) *# create meshgrid*Z = f8([X, Y]) *# Calculate Z  
  
# Plot the 3D surface for first function from project*fig = plt.figure()  
ax = fig.gca(projection=**'3d'**) *# set the 3d axes*ax.plot\_surface(X, Y, Z,  
 rstride=3,  
 cstride=3,  
 alpha=0.3,  
 cmap=**'hot'**)  
plt.show()  
  
*#Function 9*X = np.linspace(-100, 100, 100) *# points from 0..10 in the x axis*Y = np.linspace(-100, 100, 100) *# points from 0..10 in the y axis*X, Y = np.meshgrid(X, Y) *# create meshgrid*Z = f9([X, Y]) *# Calculate Z  
  
# Plot the 3D surface for first function from project*fig = plt.figure()  
ax = fig.gca(projection=**'3d'**) *# set the 3d axes*ax.plot\_surface(X, Y, Z,  
 rstride=3,  
 cstride=3,  
 alpha=0.3,  
 cmap=**'hot'**)  
plt.show()

**THIS CODE IS FOR PARTICLE SWARM OPTIMIZATION**

**import** numpy **as** np  
**import** pylab **as** py  
**from** algorithmChecker **import** \*  
**import** csv  
  
  
**class** Particle:  
 **def** \_\_init\_\_(self, dim=10):  
 **pass** self.\_\_dim = dim  
  
**class** PSO:  
 **def** \_\_init\_\_(self, func, bounds, initPos=None):  
  
 *# number of particles in swarm* self.nPart = 100  
  
 *# Control Parameters* self.epsError = 1  
 self.maxGen = 3000  
 self.w = 0.2  
 self.phiP = 0.2  
 self.phiG = 0.1  
 self.default = -1  
  
 *# Function to be minimised* self.problem = func  
  
 *# Set up boundary values* self.minBound = np.array(bounds[0])  
 self.maxBound = np.array(bounds[1])  
  
 self.dim = len(bounds[0])  
 *#Setup Dimensions  
  
  
 # Initial positions* **if** initPos!=None:  
 self.initPos = np.array(initPos).reshape((self.default,self.dim))  
 **else**:  
 self.initPos = initPos  
  
  
 **def** \_\_initPart(self):  
 *"""Initiate particles.  
 """  
  
 # Create particles* self.Particles = []  
 **for** i **in** range(self.nPart):  
 self.Particles.append( Particle(self.dim) )  
  
 *# Initiate pos and fit for particles* **for** part **in** self.Particles:  
  
 *# Initial position* **if** self.initPos == None:  
 part.pos = np.random.random(self.dim)\*self.maxBound - self.minBound  
 **else**:  
 part.pos = self.initPos[0,:]  
 self.initPos = np.delete(self.initPos, 0,0)  
  
 *# If nothing left on initial pos* **if** len(self.initPos) == 0:  
 self.initPos = None  
  
 *# Initial velocity* part.vel = np.random.random(self.dim)\*(self.maxBound - self.minBound)  
 part.vel \*= [-1, 1][np.random.random()>0.5]  
  
 *# Initial fitness* part.fitness = self.problem(part.pos)  
 part.bestFit = part.fitness  
 part.bestPos = part.pos  
  
 *# Global best fitness* self.globBestFit = self.Particles[0].fitness  
 self.globBestPos = self.Particles[0].pos  
 **for** part **in** self.Particles:  
 **if** part.fitness < self.globBestFit:  
 self.globBestFit = part.fitness  
 self.globBestPos = part.pos  
  
 **def** update(self):  
  
 **for** part **in** self.Particles:  
  
 *# Gen param* rP, rG = np.random.random(2)  
  
 w, phiP, phiG = self.w, self.phiP, self.phiG  
  
 *# Update velocity* v, pos = part.vel, part.pos  
 part.vel = self.w\*v + phiP\*rP\*(part.bestPos-pos) + phiG\*rG\*(self.globBestPos-pos)  
  
 *# New position* part.pos += part.vel  
  
 *# If pos outside bounds* **if** np.any(part.pos<self.minBound):  
 NFC = part.pos<self.minBound  
 part.pos[NFC] = self.minBound[NFC]  
 **if** np.any(part.pos>self.maxBound):  
 NFC = part.pos>self.maxBound  
 part.pos[NFC] = self.maxBound[NFC]  
  
 *# New fitness* part.fitness = self.problem(part.pos)  
  
 *# Global and local best fitness* **for** part **in** self.Particles:  
  
 *# Comparing to local best* **if** part.fitness < part.bestFit:  
 part.bestFit = part.fitness  
  
 *# Comparing to global best* **if** part.fitness < self.globBestFit:  
 self.globBestFit = part.fitness  
 self.globBestPos = part.pos  
  
 **def** optimize(self):  
 *""" Optimisation function.  
 Before it is run, initial values should be set.  
 """  
  
 # Initiate particles* self.\_\_initPart()  
 self.listOfPos = []  
  
 NFC = 0  
 **while**(NFC < self.maxGen):  
 *#print "Run: " + str(NFC) + " Best: " + str(self.globBestFit)  
  
 # Perform search* self.update()  
  
 *#Acceptably close to solution* **if** self.globBestFit < self.epsError:  
 **return** self.globBestPos, self.globBestFit  
  
 *# next gen* NFC += 1  
 self.listOfPos.append(self.globBestFit)  
 *# Search finished* **return** self.globBestPos, self.globBestFit, self.listOfPos  
  
*#################################***if** \_\_name\_\_ == **"\_\_main\_\_"**:  
*#############RUNS#######################* N = 100  
 outputFile = open(**'output9.csv'**, **'w'**)  
 outputWriter = csv.writer(outputFile)  
 outputWriter.writerow([**'Function 6'**])  
 outputWriter.writerow([**'Run'**,**'Best Fit'**, **'Best Solution'**])  
 t = np.linspace(-100, 100, N)  
 minProb = **lambda** t: f6(t)  
 numParam = 4  
 bounds = ([0]\*numParam, [10]\*numParam)  
 pso = PSO(minProb, bounds)  
 **for** i **in** range(25):  
 g = pso.optimize()  
 outputWriter.writerow([[i+1],g[0], g[1]])  
  
 *############################  
 # Visual results representation---uncomment for plotting performance* py.figure()  
 py.plot(g[2])  
 py.xlabel(**"NFC"**)  
 py.ylabel(**"Best Fit Performance"**)  
 py.title(**"PSO Performance Vs NFC"**)  
 py.show()

THIS CODE IS FOR DIFFERENTIAL EVOLUTION

**from** \_\_future\_\_ **import** division, print\_function  
**from** algorithmChecker **import** \*  
  
**import** numpy **as** np  
**from** numpy.random **import** random **as** \_random, randint **as** \_randint  
  
  
**class** DiffEvolOptimizer(object):  
  
 **def** \_\_init\_\_(self, fun, bounds, npop, F=0.8, C=0.9, seed=None, maximize=False):  
 **if** seed **is not** None:  
 np.random.seed(seed)  
  
 self.fun = fun  
 self.bounds = np.asarray(bounds)  
 self.npop = npop  
 self.F = F  
 self.C = C  
  
 self.ndim = (self.bounds).shape[0]  
 self.m = -1 **if** maximize **else** 1  
  
 bl = self.bounds[:, 0]  
 bw = self.bounds[:, 1] - self.bounds[:, 0]  
 self.population = bl[None, :] + \_random((self.npop, self.ndim)) \* bw[None, :]  
 self.fitness = np.empty(npop, dtype=float)  
 self.\_minidx = None  
  
 **def** step(self):  
 *"""Take a step in the optimization"""* rnd\_cross = \_random((self.npop, self.ndim))  
 **for** i **in** xrange(self.npop):  
 t0, t1, t2 = i, i, i  
 **while** t0 == i:  
 t0 = \_randint(self.npop)  
 **while** t1 == i **or** t1 == t0:  
 t1 = \_randint(self.npop)  
 **while** t2 == i **or** t2 == t0 **or** t2 == t1:  
 t2 = \_randint(self.npop)  
  
 v = self.population[t0,:] + self.F \* (self.population[t1,:] - self.population[t2,:])  
  
 crossover = rnd\_cross[i] <= self.C  
 u = np.where(crossover, v, self.population[i,:])  
  
 ri = \_randint(self.ndim)  
 u[ri] = v[ri]  
  
 ufit = self.m \* self.fun(u)  
  
 **if** ufit < self.fitness[i]:  
 self.population[i,:] = u  
 self.fitness[i] = ufit  
  
 @property  
 **def** value(self):  
 *"""The best-fit value of the optimized function"""* **return** self.fitness[self.\_minidx]  
  
 @property  
 **def** location(self):  
 *"""The best-fit solution"""* **return** self.population[self.\_minidx]  
  
 @property  
 **def** index(self):  
 *"""Index of the best-fit solution"""* **return** self.\_minidx  
  
  
 **def** iteroptimize(self, ngen=100):  
  
 **for** i **in** xrange(self.npop):  
 self.fitness[i] = self.m \* self.fun(self.population[i,:])  
  
 **for** j **in** xrange(ngen):  
 self.step()  
 self.\_minidx = np.argmin(self.fitness)  
 *#print("Fitness Value: " + str(self.fitness))* **yield** self.population[self.\_minidx,:], self.fitness[self.\_minidx]  
 *#print("Fitness Value: " + str(self.fitness[self.\_minidx]))* **def** \_\_call\_\_(self, ngen=1):  
 **return** self.iteroptimize(ngen)  
  
**from** de **import** DiffEvolOptimizer  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**import** csv  
  
  
*# setup the optimization*ngen, npop, ndim = 3000, 100, 10  
limits = [[-5, 5]] \* ndim  
ax = plt.subplot(2, 2, 2)  
de = DiffEvolOptimizer(f1, limits, npop)  
outputFile = open(**'output.csv'**, **'w'**)  
outputWriter = csv.writer(outputFile)  
outputWriter.writerow([**'Function 1'**])  
outputWriter.writerow([**'Run'**,**'Best Fit'**, **'Best Solution'**])  
**for** i **in** range(6):  
 g=[]  
 de.iteroptimize()  
 **print**(**"Best Fit Location: "** + str(de.location))  
 **print**(**"Best Fit Solution: "** + str(de.value))  
 g.append(str(de.location))  
 g.append(str(de.value))  
 outputWriter.writerow([[i+1],g[0], g[1]])  
*# store all the values during iterations for plotting.*pop = np.zeros([ngen, npop, ndim])  
loc = np.zeros([ngen, ndim])  
**for** i, res **in** enumerate(de(ngen)):  
 loc[i,:] = de.value.copy()  
**print**(**"Best Fit Location: "** + str(de.location))  
**print**(**"Best Fit Solution: "** + str(de.value))  
  
plt.figure()  
plt.plot(loc, **'b-'**)  
plt.title(**'DE Performance vs. NFC'**)  
plt.ylabel(**'Best fitness error'**)  
plt.xlabel(**'NFC'**)  
plt.show()