PH20105: C coding coursework

Candidate number: 23142

01/05/2020

1 - the minimum of Rosenbrock's parabolic valley (10 points)

$$y = f(x_0, x_1) = 100(x_1 - x_0^2)^2 + (1 - x_0)^2$$

Let $x_0 = z$, $x_1 = x$

$$= 100(x - z^2)^2 + (1 - z)^2$$

$$\therefore \quad f_x = 200(x - z^2) \qquad f_z = -400z(x - z^2) - 2(1 - z)$$

$$f_{xx} = 200 \qquad f_{zz} = -400(x - 3z^2) + 2$$

$$f_{xz} = -400z$$

$$f_x = 0 \ : \ 200(x - z^2) = 0 \quad \sim ①$$

$$f_z = 0 \ : \ -400z(x - z^2) - 2(1 - z) = 0 \quad \sim ②$$

$$① \to x = z^2 \Rightarrow ② : \ -400z(z^2 - z^2) - 2(1 - z) = 0$$

$$z = 1$$

$$\therefore x = 1$$

$$\therefore \text{Minimum @ } (1,1) = (x_0, x_1)$$

To verify minimum: $D = f_{xx} \cdot f_{zz} - (f_{xz})^2$

$$D = 200(-400(x - 3z^2) + 2) - (-400z)^2$$

$$D(1,1) = 400$$

$$D > 0 \text{ and } f_{xx} > 0 \ \therefore \text{ minimum}$$

$$\therefore f(1,1) = 0 \quad \therefore y = 0 \text{ @ } (1,1) \text{ minimum}$$

$$\therefore (x_0, x_1) = (1,1)$$

2 - Rosenbrock’s parabolic valley numerically (20 points)

```c
double F(double x0, double x1){
    return ( 100 * pow(x1 - x0*x0, 2.)) + pow(1 - x0, 2.);
}
```

```c
int main(){

    //Outputting function to text file

    FILE* fpointer = fopen("func_output.txt", "w");
    float x0, x1;
    for (x0 = -2; x0 <= 2; x0 += 0.05) {
        x1 = 1.;
        fprintf(fpointer, "%.2f\t%.2f\n", x0, F(x0, x1));
    }
    fclose(fpointer);
```

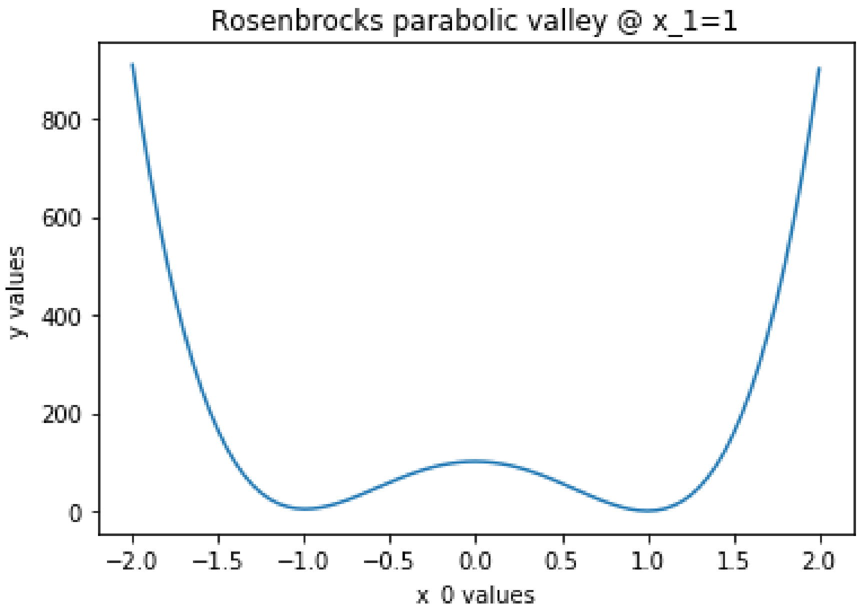
main.c | func_output.txt

```
-2.00	909.00
-1.95	794.10
-1.90	689.62
-1.85	594.97
-1.80	509.60
-1.75	432.95
-1.70	364.50
-1.65	303.72
-1.60	250.12
-1.55	203.20
-1.50	162.50
-1.45	127.55
-1.40	97.92
-1.35	73.17
-1.30	52.90
-1.25	36.70
-1.20	24.20
-1.15	15.02
-1.10	8.82
-1.05	5.25
-1.00	4.00
-0.95	4.75
-0.90	7.22
-0.85	11.12
-0.80	16.20
-0.75	22.20
-0.70	28.90
-0.65	36.07
-0.60	43.52
-0.55	51.05
-0.50	58.50
```

```python
#Python code to plot the txt file
import numpy as np
import matplotlib.pyplot as plt

data = np.loadtxt('func_output.txt')

plt.plot(data[:,0], data[:,1])
plt.title('Rosenbrocks parabolic valley @ x_1=1')
plt.xlabel('x_0 values')
plt.ylabel('y values')
plt.show()
```



The above shows the C code to output the Rosenbrocks function over the desired limits, then the corresponding textfile on the bottom left. With the python code to extract the textfile and plot.

3 Downhill simplex (70 points total)

$f(x) = 100(x_1 - x_0) + (1 - x_0) + f(x_0, x_1)$

Ref	Exp	Contr
$p^* = 2\bar{p} - P_h$	$p^{**} = 2p^* - \bar{p}$	$p^{**} = \frac{P_h + \bar{p}}{2}$

$P_i = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$ $\quad y_i = f(P_i)$

P_h and y_h = largest y_i value

P_L and y_L = lowest y_i value

Failed contraction:

$P_i = \frac{P_i + P_L}{2}$

$\bar{p}$ is centre of the 2 P_i where $i \neq h$

---

nter: $P_0 = (0,0)$ $\quad P_1 = (2,0)$ $\quad P_2 = (0,2)$

$y_0 = 1$ $\quad y_1 = 1601$ $\quad y_2 = 401$

$\therefore \underset{P_1 = P_h}{y_1 = y_h}$ and $\underset{P_0 = P_L}{y_0 = y_L}$

$\underset{\bar{y} = 101}{\bar{p} = (0,1)}$; $p^* = 2\bar{p} - P_h = (-2, 2)$; $y^* = 409$

$y^* < y_L$? No ; $y^* > y_i$? Yes ; $y^* > y_h$? No ; $P_h \leftarrow p^*$ $\therefore$ reflection

$\therefore P_h = (-2, 2)$; $y_h = 409$

$p^{**} = \frac{P_h + \bar{P}}{2} = (-1, 1.5)$; $y^{**} = 29$

contraction

$y^{**} > y_h$? No ; $p^{**} \rightarrow P_h$ $\quad \therefore P_h = (-1, 1.5)$, $y_h = 29$

Min not reached, criteria = $229 < 10^{-8}$ X.

ter:

$P_L = P_0 = (0,0)$, $\quad P_1 = (-1, 1.5)$, $\quad P_2 = (0, 2) = P_h$

$y_0 = 1 = y_L$ $\quad y_1 = 29$ $\quad y_2 = 401 = y_h$

$\bar{p} = (-0.5, 0.75)$, $\bar{y} = 27.25$; $p^* = (-1, -0.5)$, $y^* = 229$

$y^* < y_L$? No ; $y^* > y_i$? Yes ; $y^* > y_h$? No ; $p^* \rightarrow P_h$

$P_h = (-1, -0.5)$; $y_h = 229$; $p^{**} = (-0.75, 0.125)$, $y^{**} = 22.2$

$y^{**} > y_h$? No ; $p^{**} \rightarrow P_h$ $\quad \therefore P_h = (-0.75, 0.125)$, $y_h = 22.2$

Min not reached, criteria = $18.94 < 10^{-8}$ X.

First two iterations shown above, (follow each line left to right).

Final output from code:

Lines 100 and 101 explain how to switch from inputting “Any starting coordinates” to the stated starting coordinates. Simply enter the desired coordinates individually and press enter after each one.

```
Enter starting points for the triangle in the order of:
P0 = {A,B}, P1 = {C,D}, P2 = {E,K}
Click enter after each entry
0
0
2
0
0
2
Your starting values in {x0,x1,F(x0,x1):
P1={0.000000,0.000000,1.000000}
P2={2.000000,0.000000,1601.000000}
P3={0.000000,2.000000,401.000000}

The vertices, P(x0,x1,F(x0,x1)), of the smallest triangle are:
P0={0.999957,0.999913,1.870241e-09}
P1={1.000017,1.000041,4.898155e-09}
P2={1.000108,1.000217,1.194000e-08}

With the closest minimum point PL(x0,x1,F(x0,x1)) =
{0.999957,0.999913,1.870241e-09}
Taking N = 58 iterations.

These are approximately the confirmed {1,1,0} minimum point.

...Program finished with exit code 0
Press ENTER to exit console.
```

This shows the method took 58 iterations to reach the criteria, well before the N=1000 criteria.

Source Code Appendix:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//Functions below

double F(double x0, double x1){
    return ( 100 * pow(x1 - x0*x0, 2.)) + pow(1 - x0, 2.);
}

double max(double N1[3][3]){          //Function to order the points from min to max F(x0,x1) in a matrix
    int n=3,i,j;
    double a,b,c,d,e,f;
    double N[3][3];
    for (i=0;i<n;i++){
        for(j=i+1;j<n;j++){
            if(N1[i][2] > N1[j][2]){
                a =  N1[i][2];
                N1[i][2] = N1[j][2];
                N1[j][2] = a;
                b =  N1[i][2];
                c =  N1[i][1];
                N1[i][1] = N1[j][1];
                N1[j][1] = c;
                d =  N1[i][1];
                e =  N1[i][0];
                N1[i][0] = N1[j][0];
                N1[j][0] = e;
                f =  N1[i][0];
            }
        }
    }
}

void replace(double Pa[], double Pb[]){          //Replace Function
    Pa[0] = Pb[0]; Pa[1] = Pb[1]; Pa[2] = Pb[2];
}

double SD(double P1[], double P2[], double P3[], double midP[]){     //Standard deviation Function
    int D = 2;  //D = no.dimensions
    return sqrt( ( pow((P1[2]-midP[2]),2) + pow((P2[2]-midP[2]),2) + pow((P3[2]-midP[2]),2) ) / D);
}

//For the five functions below, there are only two operations occuring
//and they could be condensed into just the two functions for simplicity
//but for ease of identification during main loop
//they have been left as individual named functions.

void midp(double P1[],double P2[],double P3[]){     //Midpoint Function
    int i;
    for(i=0;i<2;i++){
        P3[i] = ((P1[i] + P2[i])/2);
    }
}

void ref(double P1[],double P2[],double P3[]){    //Reflection Function
    int i;
    for(i=0;i<2;i++){
        P3[i] = (2*P1[i] - P2[i]);
    }
}

void expa(double P1[],double P2[],double P3[]){    //Expansion Function
    int i;
    for(i=0;i<2;i++){
        P3[i] = (2*P1[i] - P2[i]);
    }
}

void cont(double P1[],double P2[],double P3[]){    //Contraction Function
    int i;
    for(i=0;i<2;i++){
        P3[i] = ((P1[i] + P2[i])/2);
    }
}

void fcont(double P1[],double P2[],double P3[]){  //Failed Contraction Function
    int i;
    for(i=0;i<2;i++){
        P3[i] = ((P1[i] + P2[i]))/2;
    }
}
```

```c
84
85  //Main Section
86
87
88  int main(){
89
90      //Outputting function to text file
91
92  //  FILE* fpointer = fopen("func_output.txt", "w");
93  //      float x0, x1;
94  //      for (x0 = -2; x0 <= 2; x0 += 0.05) {
95  //          x1 = 1.;
96  //          fprintf(fpointer, "%.2f\t%.2f\n", x0, F(x0, x1));
97  //      }
98  //      fclose(fpointer);
99
100     //Lines 102-113 are for input of any starting points, feel free
101     //to comment out 102-113 and uncomment lines 116-118 instead.
102
103     float A,B,C,D,E,K;      //F was used in Rosenbrock function so unfortunetly the alphabet will skip a letter.
104     double P0[3];
105     double P1[3];
106     double P2[3];
107     printf("\033[1;35m");
108     printf("Enter starting points for the triangle in the order of:");
109     printf("\nP0 = {A,B}, P1 = {C,D}, P2 = {E,K}\n");
110     printf("Click enter after each entry\n");
111     scanf("\n%f",&A); scanf("\n%f",&B); scanf("\n%f",&C); scanf("\n%f",&D); scanf("\n%f",&E); scanf("\n%f",&K);
112     P0[0] = A; P0[1] = B; P1[0] = C; P1[1] = D; P2[0] = E; P2[1] = K;
113     P0[2] = F(A,B); P1[2] = F(C,D); P2[2] = F(E,K);
114     printf("Your starting values in {x0,x1,F(x0,x1):\nP1={%f,%f,%f}\nP2={%f,%f,%f}\nP3={%f,%f,%f}\n",
115     P0[0],P0[1],P0[2],P1[0],P1[1],P1[2],P2[0],P2[1],P2[2]);
116     printf("\033[0m");
117
118     //Standard starting points
119
120 //  double P0[3] = {0,0,F(0,0)};
121 //  double P1[3] = {2,0,F(2,0)};
122 //  double P2[3] = {0,2,F(0,2)};
123     double PM[3];   //P-bar, the midpoint
124     double Pst[3];  //P*, P_h upon reflection
125     double Pstst[3];    //P**, P_h upon expansion or contraction
126     int N = 0;      //Counter for iterations
127
128     //Flow-chart
129
130     while( SD(P0,P1,P2,PM) > pow(10, -8)){
131
132         double A[3][3] = { {P0[0],P0[1],P0[2]}, {P1[0],P1[1],P1[2]}, {P2[0],P2[1],P2[2]} };
133         max(A);     //Has ordered points in ascending F(x0,x1)
134
135         double PL[3] = {A[0][0],A[0][1],A[0][2]};   //Extracts the, now, ordered arrays based
136         double Pi[3] = {A[1][0],A[1][1],A[1][2]};   //on their third value - F(x0,x1)
137         double PH[3] = {A[2][0],A[2][1],A[2][2]};
138
139         midp(PL,Pi,PM);     //Identifies midpoint of P_low and P_i
140         PM[2] = F(PM[0],PM[1]);     //F(x0,x1) for the midpoint coords.
141
142         ref(PM, PH, Pst);       //Indentifies reflection point as Pst.
143         Pst[2] = F(Pst[0],Pst[1]);
144
145         if( Pst[2] < PL[2] ){       //First block in flow-chart
146             expa(Pst,PM,Pstst);         //Expansion
147             Pstst[2] = F(Pstst[0],Pstst[1]);
148                 if( Pstst[2] < PL[2] ){
149                     replace(PH,Pstst);      //Replacement
150                     replace(P2,Pstst);
151                     replace(P0,PL);
152                     replace(P1,Pi);
153                     P0[2] = F(P0[0],P0[1]);     //Fills in the F(x0,x1) for each point as
154                     P1[2] = F(P1[0],P1[1]);     //the "replace" function swaps cells 0 and 1,
155                     P2[2] = F(P2[0],P2[1]);     //after this, cell 2 must be calculated.
156                 }
157                 else{
158                     replace(PH,Pst);        //Replacement
159                     replace(P2,Pst);
160                     replace(P0,PL);
161                     replace(P1,Pi);
162                     P0[2] = F(P0[0],P0[1]);
163                     P1[2] = F(P1[0],P1[1]);
164                     P2[2] = F(P2[0],P2[1]);
165                 }
166         }
```

```c
        else{
            if( Pst[2] > Pi[2]){
                if( Pst[2] > PH[2]){
                    cont(PH,PM,Pstst);      //Contraction
                    Pstst[2] = F(Pstst[0],Pstst[1]);
                    if( Pstst[2] > PH[2]){
                        fcont(P0,PL,P0);        //Failed Contraction
                        fcont(P1,PL,P1);
                        fcont(P2,PL,P2);
                        P0[2] = F(P0[0],P0[1]);
                        P1[2] = F(P1[0],P1[1]);
                        P2[2] = F(P2[0],P2[1]);
                    }
                    else{
                        replace(PH,Pstst);      //Replacement
                        replace(P2,Pstst);
                        replace(P0,PL);
                        replace(P1,Pi);
                        P0[2] = F(P0[0],P0[1]);
                        P1[2] = F(P1[0],P1[1]);
                        P2[2] = F(P2[0],P2[1]);
                    }
                }
                else{
                    replace(PH,Pst);        //Replacement followed by Contraction
                    cont(PH,PM,Pstst);
                    Pstst[2] = F(Pstst[0],Pstst[1]);
                        if( Pstst[2] > PH[2] ){
                            fcont(P0,PL,P0);        //Failed Contraction
                            fcont(P1,PL,P1);
                            fcont(P2,PL,P2);
                            P0[2] = F(P0[0],P0[1]);
                            P1[2] = F(P1[0],P1[1]);
                            P2[2] = F(P2[0],P2[1]);
                        }
                        else{
                            replace(PH,Pstst);      //Replacement
                            replace(P2,Pstst);
                            replace(P0,PL);
                            replace(P1,Pi);
                            P0[2] = F(P0[0],P0[1]);
                            P1[2] = F(P1[0],P1[1]);
                            P2[2] = F(P2[0],P2[1]);
                        }
                        }
                }
            }
            else{
                replace(PH,Pst);        //Replacement
                replace(P2,Pst);
                replace(P0,PL);
                replace(P1,Pi);
                P0[2] = F(P0[0],P0[1]);
                P1[2] = F(P1[0],P1[1]);
                P2[2] = F(P2[0],P2[1]);
            }
        }

        N = N + 1;      //Counter
        if(N == 1000){      //N=1000 break criteria
            break;
        }
    }

    double X[3][3] = { {P0[0],P0[1],P0[2]}, {P1[0],P1[1],P1[2]}, {P2[0],P2[1],P2[2]} }; //Orders points again
    max(X);     //Has ordered points in ascending F(x0,x1)

    printf("\033[1;36m");       //some colour for fun
    printf("\nThe vertices, P(x0,x1,F(x0,x1)), of the smallest triangle are:\nP0={%f,%f,%e}\nP1={%f,%f,%e}\nP2={%f,%f,%e}\n",
    X[0][0],X[0][1],X[0][2],X[1][0],X[1][1],X[1][2],X[2][0],X[2][1],X[2][2]);
    printf("\nWith the closest minimum point PL(x0,x1,F(x0,x1)) = \n{%f,%f,%e}\n", X[0][0],X[0][1],X[0][2]);
    printf("Taking N = %d", N);       //N-1 because printf statement is outside the loop, so it counts an extra iteration
    printf(" iterations.\n");
    printf("\nThese are approximately the confirmed {1,1,0} minimum point.");
    printf("\033[0m");

    return(0);
}
```