Candidate number: 24563

PH20105 Hand-In-Exercise 2022-23

1-The minimum of the Rosenbrock valley

Logic:

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

y is composed of two squares and therefore $y \geq 0$

Assuming the minimum occurs at $y = 0$ => $x_1 - x_0^2 = 0$ and $1 - x_0 = 0$

By substituting and solving for x_1 and x_0 we find that the minimum occurs at $F(1,1)$

$x_0 = 1$ => $x_1 - (1)^2 = 0$ => $x_1 = 1$

Differentiating:

$$\frac{\partial y}{\partial x_0} = -400x_0(x_1 - x_0^2) - 2(1 - x_0)$$

$$\frac{\partial y}{\partial x_1} = 200(x_1 - x_0^2)$$

$$\frac{\partial y}{\partial x_0} = \frac{\partial y}{\partial x_1} = 0$$

From the second equation => $x_1 = x_0^2$

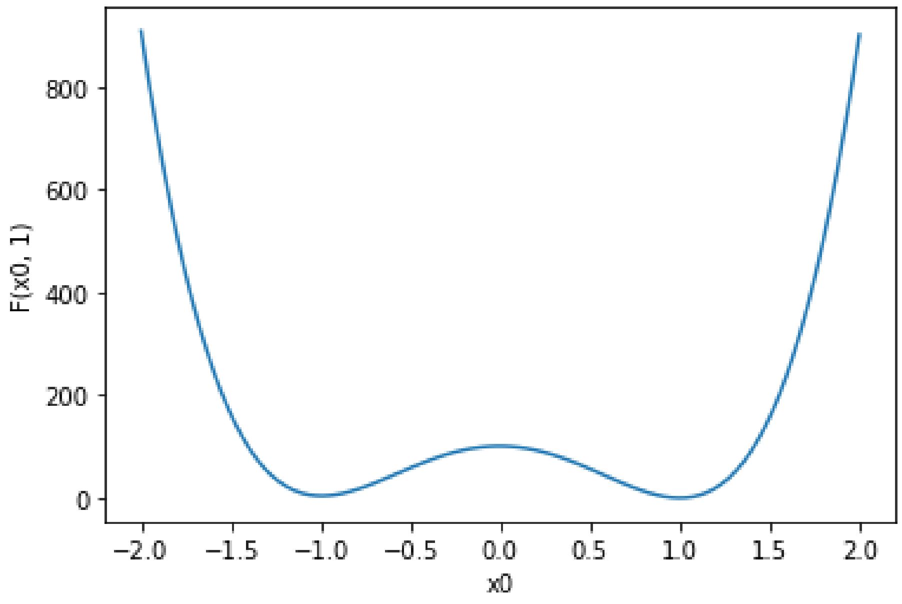
Subbing into the first equation => $x_0 = 1$ and therefore $x_1 = \pm 1$

Subbing $x_1 = -1$ into the original equation we get $y = 400$

Subbing $x_1 = 1$ into the original equation we get $y = 0$

Therefore we can conclude the minimum occurs at $F(1,1)$, we know this is a minimum and not a saddle point since y cannot take any values lower than zero.

2-Rosenbrock’s parabolic valley numerically



3-Downhill simplex

1st iteration:

$P_0 = \binom{0}{0},\ P_1 = \binom{2}{0},\ P_2 = \binom{0}{2}$

$y_0 = 1,\ y_1 = 1601,\ y_2 = 401$ => $y_0 = y_l,\ y_1 = y_h,\ y_2 = y_i$ and $P_0 = P_l,\ P_1 = P_h,\ P_2 = P_i$

$\overline{P} = \frac{P_i + P_l}{2} = \binom{0}{1}$

$P^* = 2\overline{P} - P_h = \binom{-2}{2}$ => $y^* = 409$

$y^* > y_i$ => $y^* < y_h$

$P^{**} = \frac{P^* + \overline{P}}{2} = \binom{-1}{1.5}$ => $y^{**} = 29$

$y^{**} < y^*$

$P_h = P^{**} = \binom{-1}{1.5} = P_1$ => $P_0 = \binom{0}{0},\ P_1 = \binom{-1}{1.5},\ P_2 = \binom{0}{2}$

$n = 2,\ \sqrt{\sum_i \frac{(y_i - \overline{y})^2}{n}} \approx 832.7$

$832.7 > 10^{-8}$

2nd iteration:

$P_0 = \binom{0}{0},\ P_1 = \binom{-1}{1.5},\ P_2 = \binom{0}{2}$

$y_0 = 1,\ y_1 = 29,\ y_2 = 401$ => $y_0 = y_l,\ y_1 = y_i,\ y_2 = y_h$ and $P_0 = P_l,\ P_1 = P_i,\ P_2 = P_h$

$\overline{P} = \frac{P_i + P_l}{2} = \binom{-0.5}{0.75}$

$P^* = 2\overline{P} - P_h = \binom{-1}{-0.5}$ => $y^* = 229$

$y^* > y_i$ => $y^* < y_h$

$P^{**} = \frac{P^* + \overline{P}}{2} = \binom{-0.75}{0.125}$ => $y^{**} = \frac{1421}{64} \approx 22.2$

$y^{**} < y^*$

$P_h = P^{**} = \binom{-0.75}{0.125} = P_2$ => $P_0 = \binom{0}{0},\ P_1 = \binom{-1}{1.5},\ P_2 = \binom{-0.75}{0.125}$

$n = 2,\ \sqrt{\sum_i \frac{(y_i - \overline{y})^2}{n}} \approx 223.3$

$223.3 > 10^{-8}$

Coordinates at the end of the 2nd iteration:

$P_0 = \binom{0}{0},\ P_1 = \binom{-1}{1.5},\ P_2 = \binom{-0.75}{0.125}$

$y_0 = 1,\ y_1 = 29,\ y_2 \approx 22.2$

Comparing to the code:

```
Final vertices:
(0.000000, 0.000000)
(-1.000000, 1.500000)
(-0.750000, 0.125000)

Evaluations:
1.000000
29.000000
22.203125

Iterations: 2
```

These coordinates match with those calculated previously.

The full code returns:

```
Final vertices:
(1.000108, 1.000217)
(0.999851, 0.999698)
(1.000017, 1.000041)

Evaluations:
0.000000
0.000000
0.000000

Iterations: 57
```

Appendix:

2-Rosenbrock’s parabolic valley numerically

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

#define N 2 //number of dimensions

typedef struct {
    double x[N];        //struct containing the coordinates of the vertices
} Point;

double func(Point P) {
    return 100 * pow(P.x[1] - pow(P.x[0], 2), 2) + pow(1 - P.x[0], 2);  //Rosenbrock function


void file_data(char *textfile, double xlow, double xhigh, int values) {

    double x0 = xlow;
    double step = (xhigh - xlow) / (values - 1);

    FILE *fp = fopen(textfile, "w");

    for (int i = 0; i < 100; i++, x0 += step) {
        Point P = {x0, 1};
        fprintf(fp, "%.3lf, %.3lf\n", x0, func(P));     //writing data into textfile
    }

    fclose(fp);
}

int main() {
    file_data("data.txt", -2., 2., 100);

    return 0;
}
```

Python:

```python
import matplotlib.pyplot as plt
import numpy as np
data = np.loadtxt("data.txt", delimiter = ', ')
x = np.array(data[:,0])
F = np.array(data[:,1])
plt.plot(x, F)
plt.xlabel("x0")
plt.ylabel("F(x0, 1)")
plt.draw()
plt.show()
```

3-Downhill simplex

```c
/*****************************************************
 *                                                   *
 * Candidate number: 24563                           *
 *                                                   *
 * Downhill Simplex algorithm for finding the        *
 * minimum of the Rosenbrock function                *
 *                                                   *
 * Starting vertices:                                *
 * p0 = (0,0), p1 = (2,0), p2 = (0,2)                *
 *                                                   *
 * This code can be amended for functions with more  *
 * variables by changing N to the number of          *
 * variables in the new function and the number of   *
 * initial vertices to N + 1                         *
 *                                                   *
 *****************************************************/

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

#define N 2     //number of dimensions

#define ALPHA 1.0
#define BETA 0.5
#define GAMMA 2.0
#define RHO 0.5     //coefficients for reflection, contraction, expansion and shrinking

#define TOL 1e-8
#define MAX_ITER 1000   //standard deviation tolerance and maximum iterations

typedef struct {
    double x[N];        //struct containing the coordinates of the vertices
} Point;

double func(Point P) {
    return 100 * pow(P.x[1] - pow(P.x[0], 2), 2) + pow(1 - P.x[0], 2);  //Rosenbrock function
}

void sortPoints(double Y[], Point P[]) {
    double temp;
    Point pTemp;
    for (int i = 0; i < N; i++) {
        for (int j = i + 1; j < N + 1; j++) {
            if (Y[i] > Y[j]) {
                temp = Y[i];
                Y[i] = Y[j];    //reordering outputs from smallest to largest
                Y[j] = temp;
                pTemp = P[i];
                P[i] = P[j];    //reordering the corresponding vertices
                P[j] = pTemp;
            }
        }
    }
}

void centroid(Point P[], Point* Pbar) {
    for (int i = 0; i < N; i++) {
        Pbar->x[i] = 0;
        for (int j = 0; j < N; j++) {
            Pbar->x[i] += P[j].x[i] / N;    //the centroid is the average of all vertices except
        }                                   //Ph (worst vertex), denoted by Pbar
    }
}

void reflect(Point* Ps, Point* Pbar, Point P[]) {
    for (int i = 0; i < N; i++) {
        Ps->x[i] = Pbar->x[i] + ALPHA * (Pbar->x[i] - P[N].x[i]);   //expansion of Pbar away from
    }                                                               //Ph, denoted by P*
}

void insideContract(Point* Pss, Point P[], Point* Pbar) {
    for (int i = 0; i < N; i++) {
        Pss->x[i] = Pbar->x[i] + BETA * (P[N].x[i] - Pbar->x[i]);   //contraction of Ph towards
    }                                                               //Pbar, denoted by P**
}

void outsideContract(Point* Pss, Point* Ps, Point* Pbar) {
    for (int i = 0; i < N; i++) {
        Pss->x[i] = Pbar->x[i] + BETA * (Ps->x[i] - Pbar->x[i]);    //contraction of P* towards
    }                                                               //Pbar, denoted by P**
}

void expand(Point* Pss, Point* Ps, Point* Pbar) {
    for (int i = 0; i < N; i++) {
        Pss->x[i] = Pbar->x[i] + GAMMA * (Ps->x[i] - Pbar->x[i]);   //expansion of P* away from
    }                                                               //Pbar, denoted by P**
}

void shrink(Point P[]) {
    for (int i = 0; i < N + 1; i++) {
        for (int j = 0; j < N; j++) {
            P[i].x[j] = P[0].x[j] + RHO * (P[i].x[j] - P[0].x[j]);  //contraction of every point
        }                                                           //towards Pl (best vertex)
    }
}

void replacePoint(Point* new, Point* orig) {
    *new = *orig;                               //replacing Ph with P* or P**
}
```

```c
_Bool MinCon(double Y[]) {
    double Ybar = 0, sum = 0;
    for (int i = 0; i < N + 1; i++) {
        Ybar += Y[i] / (N + 1);
    }
    for (int i = 0; i < N + 1; i++) {
        sum += pow(Y[i] - Ybar, 2) / N;
    }
    return (sqrt(sum) < TOL);   //condition for breaking the loop containing the simplex method
}

void simplex(Point P[]) {
    Point Pbar, Ps, Pss;
    double Ys, Yss, Y[N + 1];

    int a;  //iterations

    for (a = 0; a < MAX_ITER; a++) {

        for (int i = 0; i < N + 1; i++) {
            Y[i] = func(P[i]);              //initialising function outputs
        }

        sortPoints(Y, P);
        centroid(P, &Pbar);
        reflect(&Ps, &Pbar, P);

        Ys = func(Ps);

        if (Ys >= Y[0] && Ys < Y[N - 1]) {      //if y* >= yl and y* <= second worst vertex
            replacePoint(&P[N], &Ps);       //replacing Ph with P*
        }

        else if (Ys < Y[0]) {           //if y* < yl
            expand(&Pss, &Ps, &Pbar);
            Yss = func(Pss);

            if (Yss < Ys) {                 //if y** < y*
                replacePoint(&P[N], &Pss);  //replacing Ph with P**
            }

            else {
                replacePoint(&P[N], &Ps);   //replacing Ph with P*
            }
        }

        else {      //if y* >= second worst vertex

            if (Ys < Y[N]) {                            //if y* < yh
                outsideContract(&Pss, &Ps, &Pbar);
                Yss = func(Pss);

                if (Yss < Ys) {                 //if y** < y*
```

```
                    replacePoint(&P[N], &Pss);      //replacing Ph with P**
                }

                else {
                    shrink(P);
                }
            }

            else {      //if y* >= yh
                insideContract(&Pss, P, &Pbar);
                Yss = func(Pss);

                if (Yss < Y[N]) {               //if y** < yh
                    replacePoint(&P[N], &Pss);      //replacing Ph with P**
                }

                else {
                    shrink(P);
                }
            }
        }

        if (MinCon(Y)) {
            break;
        }
    }

    printf("Final vertices:\n");
    for (int i = 0; i < N + 1; i++) {
        printf("(%lf, %lf)\n", P[i].x[0], P[i].x[1]);
    }

    printf("\nEvaluations:\n");
    for (int i = 0; i < N + 1; i++) {
        printf("%lf\n", func(P[i]));
    }

    printf("\nIterations: %d\n", a + 1);
}

int main() {
    Point P[N + 1] = { {0, 0}, {2, 0}, {0, 2} };    //initial vertices (each vertex has N coordinates)
    simplex(P);
    return 0;
}
```