מעבדה באופטימיזציה

פרויקט מסכם

בתרגיל זה נסב את התוכנית steepest_golden7.c כך שהתוכנית תמצא בעצמה את וקטור ההערכה ההתחלתית x0 לפתרון וכן ערך איכותי של אפסילון, רמת הדיוק של הנורמה הרב ממדית. כהערת אגב נאמר כאן שמדובר כאן בניסיון להתמודד עם שני בעיות רציניות בכל הקשור לתכנות אלגוריתמים כאלו.

אנחנו נצא מתוך הנחה (שוב מגבלה של סוג זה של תכנות) שידוע לנו טווח ערכים ***(L,H)*** של המשתנים שאנחנו מחפשים). זו הערכה סבירה במובן הזה שבדרך כלל אפשר לעשות scaling של הנוסחאות כך שנוכל לעבוד עם ערכי משתנים בטווח נניח (-10.0, 10.0) ולא נצטרך להתמודד עם מספרים בסדר גודל של 10^{38}.

בתרגיל הזה נממש מציאת x0 התחלתי ע"י רעיון דומה ל-Coordinate Descent:

יש לנו בעיה רב ממדית. במימד אחד אפשר לצעוד בכיוון שהפונקציה יורדת עד שנמצא שלישיה ***f(x) > f(x+h), f(x+h) <f(x + 2h)*** ואז ***x+h*** יכול לשמש כקירוב ל-***x0***. בממד אחד הרעיון הזה די אמין אבל כאן אנחנו בבעיה רב ממדית וזה פשוט להכליל את זה. דרך אחת שעשויה להועיל הוא ***מקדם אחר מקדם***:

מתחילים עם

mid = (L+H)/2

xm= (L, mid, mid, ..., mid)

מחפשים שלישיה

x0jm= (L+jh, mid, mid, ..., mid)

x0j1m = (L+(j+1)h, mid, mid, ..., mid)

x0j2m= (L+(j+2)h, mid, mid, ..., mid)

המקיימים

f(x0jm) > f(x0j1m), f(x0j1m) < f(x0j2m)

ואז ***x00h = L+(j+1)h*** יהיה קירוב ראשוני ל-***x0[0].***

חוזרים על התהליך הזה עבור ***x1*** עם נקודת התחלה

xm= (x00h, L, mid, ..., mid)

וכן הלאה עד ל-n-1 עם נקודות התחלה:

xm= (x00h, x01h, L, mid, ..., mid)

xm= (x00h, x01h, x02h, L, mid, ..., mid)

.....

xm= (x00h, x01h, x02h, x03h, ..., L)

לאחר חוזרים על אותו תהליך עם נקודות התחלה:

xm= (L, x01h, x02h, x03h, ..., x0n-1h)

xm= (x00h*, L, x01h, x02h, x03h, ..., x0n-1h)

xm= (x00h*, x01h*, L, x02h, x03h, x04h, ..., x0n-1h)

...

xm= (x00h*, x01h*, x01h*, x02h*, x03h*, ..., L)

כאשר * מציין ערכים שחושבו בסיבוב השני.

הווקטור ***xm*** הסופי ישמש ערך התחלתי x0 עבור steepest_decsent עם חיפוש קווי יחס הזהב עם ערך מקורי של אפסילון.

את steepest_decent יש לחזור ולבצע עם ערך ***x0*** מאותחל עם התוצאה ***xn*** של השלב הקודם עם ערך אפסילון שהוא חצי מהאפסילון של השלב הקודם, עד אשר המנה בין סכום ווקטור ההפרשים בערך מוחלט בין שתי ההרצות האחרונות של steepest יהיה גדול מנניח 0.995.

הכותרת של הסכמה אותה צריך לממש הינו:

```
void mutli_variable_optimization_schema(
double xn[],  int n,
FUN_PTR f, GRAD_FUN_PTR grad, double epsilon,
```

VECTOR_CONVERGENCE_TEST v,
double lowb, double highb, int initial_m)

כאשר lowb, highb הם החסמים העליון והתחתון של הערכים של המקדמים, xn הוא וקטור היעד לתוצאה. N הוא המימד של הוקטור xn, הממד של בעיית האופטימיזציה. f הוא פוינטר לפונקציה לפונקציית המטרה, epsilon תהיה הערכה ראשונית לדיוק, v יהיה (כמו קודם) פוינטר לפונקציה לחישוב בדיקת נורמה של וקטור לעומת אפסילון, ו-initial_m יהיה מספר החלקים שיש לחלק את אינטרוול חיפוש ה-x0 הראשוני.

לדוגמא, הפלטים של התוכנית הבאה:

```
double f(double x[])
{
 int i, j;
 double sum;

 // return (-sin(x[0]+2*x[1]) - cos(3*x[0]+4*x[1]));
 vector_matrix_mult(ftemp_arr, x, Q,  3);
 sum = 0.5 * mult_vector_vector(ftemp_arr, x, 3);
 sum = sum + mult_vector_vector(b, x, 3);

return sum;

} // f

int main()
{
 double xstar[10], x0[10], value;
 double qarr[16] = {2, -1, 0, -1, 2, -1, 0, -1,  2};
 int i, j;


for(i=0; i < 3; i++)
 for(j=0; j < 3; j++)
        Q[i][j] = qarr[i*3 + j];

printf("Q: \n");
for(i=0; i < 3; i++)
{
```

```
  for(j=0; j < 3; j++)
    printf(" %6.2lf ", Q[i][j]);
  printf("\n");
} // for

b[0] = -1;
b[1] = -2;
b[2] = -3;

mutli_variable_optimization_schema(xstar, x0, 3,
   f, approx_g, 0.00001, vector_convergence_test,
   0.0, 10.0,  100);


  printf("\n\noptimal solution:\n xstar[0] = %lf\n,  xstar[1] = %lf,  xstar[2] = %lf\n",
           xstar[0], xstar[1],  xstar[2] );

  printf("\n\nIn degrees: xstar[0] = %lf\n,  xstar[1] = %lf\n",
           xstar[0]*180.0/M_PI, xstar[1]*180.0/M_PI);

  printf("\n\noptimal value  = %lf\n", f(xstar));


} // main
```

יכול להיות משהו כמו:

```
optimal solution:
 xstar[0] = 2.500017
,  xstar[1] = 4.000078,  xstar[2] = 3.500017


In degrees: xstar[0] = 143.240444
,  xstar[1] = 229.187572


optimal value  = -10.500000
```

פלט של התוכנית הבאה:

```
double f(double x[])
{
  int i, j;
  double sum;

 return (-sin(x[0]+3*x[1] + 15*x[2]) - sin(3*x[0]+3*x[1] + 9*x[2]) -
sin(x[0]+x[1] + 18*x[2]) );

return sum;

} // f

int main()
{
  double xstar[10],  value;
  int i, j;


mutli_variable_optimization_schema(xstar,  3,
   f, approx_g, 0.00001, vector_convergence_test,
   0.0, 1.0,  100);


  printf("\n\noptimal solution:\n xstar[0] = %lf\n,  xstar[1] = %lf,
xstar[2] = %lf\n",
            xstar[0], xstar[1],  xstar[2] );

  printf("\n\nIn degrees: xstar[0] = %lf\n,  xstar[1] = %lf\n,  xstar[2] =
%lf\n",
            xstar[0]*180.0/M_PI, xstar[1]*180.0/M_PI,
xstar[2]*180.0/M_PI);

  printf("\n\noptimal value  = %lf\n", f(xstar));
```

```
} // main
```

יכול להחיות כמו:

```
optimal solution:
 xstar[0] = 0.208690
,  xstar[1] = 0.105402,  xstar[2] = 0.069788


In degrees: xstar[0] = 11.957033
,  xstar[1] = 6.039075
,  xstar[2] = 3.998581


optimal value  = -2.999999
```

סביר להניח שלא בהכרח תקבלו בדיוק אותם ערכים, מספיק שיהיו דומים.