บทที่ 3

การสร้างซอฟต์แวร์

ในบทนี้จะกล่าวถึงรายละเอียดของการสร้างซอฟต์แวร์เพื่อหาผลเฉลยเหมาะที่สุดของปัญหากำหนดการไม่เชิงเส้นภายใต้เงื่อนไขบังคับเชิงเส้น รวมทั้งขั้นตอนการออกแบบซอฟต์แวร์ประกอบด้วยผังงานของซอฟต์แวร์และขั้นตอนวิธีของซอฟต์แวร์ การสร้างซอฟต์แวร์ในวิทยานิพนธ์นี้ได้เขียนโปรแกรมภาษา C เพราะว่าเป็นโปรแกรมภาษาที่เหมาะต่องานทางด้านการคำนวณ รวมทั้งซอฟต์แวร์ที่ช่วยคำนวณในงานวิจัยนี้คือ ADOL-C คำนวณหาค่าเกรเดียนต์ของฟังก์ชันจุดประสงค์ และ GLPK หาผลเฉลยของปัญหากำหนดการเชิงเส้น สามารถเขียนคำสั่งการคำนวณเป็นภาษา C ได้ และเนื้อหาสุดท้ายของบทนี้เป็นเรื่องวิธีการใช้งานซอฟต์แวร์ ซึ่งมีรายละเอียดในแต่ละหัวข้อดังนี้

3.1 Automatic differentiation

นิยาม ฟังก์ชันมูลฐาน (Elementary functions) คือ ฟังก์ชันที่อยู่ในรูปของ

ฟังก์ชันค่าคงที่ (Constant function)

ฟังก์ชันกำลัง (Power function)

ฟังก์ชันเลขชี้กำลัง (Exponential function)

ฟังก์ชันลอการิทึม (Logarithmic function)

ฟังก์ชันตรีโกณมิติ (Trigonometric function)

ฟังก์ชันตรีโกณมิติผกผัน (Inverse trigonometric function)

ฟังก์ชันไฮเพอร์โบลิก (Hyperbolic function)

ฟังก์ชันไฮเพอร์โบลิกผกผัน (Inverse hyperbolic function)

การหาอนุพันธ์ของฟังก์ชันมูลฐานไม่ซับซ้อนเป็นการหาอนุพันธ์ที่ทราบสูตรชัดเจนสามารถเขียนได้โดยง่าย แต่ถ้าฟังก์ชันจุดประสงค์ของปัญหากำหนดการไม่เชิงเส้นไม่เป็นฟังก์ชันมูลฐาน การหาค่าอนุพันธ์ตามสูตรจะยุ่งยากและซับซ้อนในงานวิจัยนี้ได้นำหลักการ Automatic differentiation [6, 8, 13] ซึ่งเป็นวิธีการหาค่าอนุพันธ์ของฟังก์ชันที่ซับซ้อนมาหาค่าอนุพันธ์ของฟังก์ชันจุดประสงค์ โดยอาศัยการแยกฟังก์ชันที่ซับซ้อนที่ต้องการหาค่าอนุพันธ์ออกเป็นฟังก์ชันมูลฐานที่มีสูตรทางแคลคูลัสชัดเจน เมื่อนำอนุพันธ์ของฟังก์ชันมูลฐานมารวมกันโดยอาศัยกฎลูกโซ่ได้ผลลัพธ์เป็นค่าอนุพันธ์ของฟังก์ชันซึ่งเป็นค่าอนุพันธ์ที่แท้จริงที่ไม่ใช่การประมาณค่า

หลักการของ Automatic differentiation มี 2 รูปแบบ คือ forward mode และ reverse mode ซึ่งจะได้อธิบายตามลำดับต่อไป

3.1.1 วิธี forward mode

ในการหาผลเฉลยของปัญหากำหนดการไม่เชิงเส้นมีการคำนวณหาค่าเกรเดียนต์ของฟังก์ชันมีหลักการตามวิธี forward mode ดังนี้ ต้องการหาค่าเกรเดียนต์ของ $f(x_1, x_2, ..., x_n)$ ซึ่งเป็นฟังก์ชันของตัวแปรอิสระ n ตัวและฟังก์ชัน f ไม่เป็นฟังก์ชันมูลฐาน

เริ่มต้นด้วยการแยกฟังก์ชันที่ซับซ้อน $f(x_1, x_2, ..., x_n)$ ออกเป็นฟังก์ชันมูลฐาน โดยการกำหนดฟังก์ชันมูลฐานแต่ละฟังก์ชันที่ประกอบเป็น f ออกมาทีละฟังก์ชัน

กำหนดให้ $i = n+1$

กำหนดตัวแปรระหว่างกลาง (intermediate variable) x_i ให้มีเท่ากับฟังก์ชันมูลฐานที่แยกออกมา นั่นคือ

$$x_i = f_i(x_k, k \in J_i)$$

เมื่อ J_i คือเซตดรรชนีที่บอกว่า x_k เป็นตัวแปรอิสระของฟังก์ชันมูลฐาน f_i

เมื่อแยกฟังก์ชันมูลฐานออกมาแล้วใช้กฎลูกโซ่หาค่าเกรเดียนต์ของฟังก์ชันมูลฐานนั้น

จะได้ว่า $\nabla x_i = \sum_{j \in J_i} \frac{\partial f_i(x_k, k \in J_i)}{\partial x_j} \nabla x_j$ เมื่อ $\nabla x_j = e_j$ คือเวกเตอร์หนึ่งหน่วยในแนวแกน x_j

กำหนดให้ $i = n+2$

กำหนดตัวแปรระหว่างกลาง x_i ให้มีค่าเท่ากับฟังก์ชันมูลฐาน นั่นคือ $x_i = f_i(x_k, k \in J_i)$

โดยกฎลูกโซ่ $\nabla x_i = \sum_{j \in J_i} \frac{\partial f_i(x_k, k \in J_i)}{\partial x_j} \nabla x_j$

เพิ่มค่า i กระทำในลักษณะแยกฟังก์ชันมูลฐานและการบวก ลบ คูณ หาร ของตัวแปรระหว่างกลาง จนกระทั่ง ตัวแปรระหว่างกลางสุดท้ายคือฟังก์ชันที่ต้องการหาค่าเกรเดียนต์ f

สมมติว่า $i = m$ ได้ $\quad x_i = f_i(x_k, k \in J_i) = f(x_1, x_2, ..., x_n)$

นั่นคือ $\quad x_m = f_m(x_k, k \in J_m) = f(x_1, x_2, ..., x_n)$

เมื่อ m คือจำนวนเต็มบวกที่ทำให้ตัวแปรระหว่างกลาง $x_m = f(x_1, x_2, ..., x_n)$

เพราะฉะนั้น สรุปได้ว่า ค่าเกรเดียนต์ของฟังก์ชัน $f(x_1, x_2, ..., x_n)$ คือ

$$\nabla f = \nabla x_m = \sum_{j \in J_m} \frac{\partial f_m(x_k, k \in J_m)}{\partial x_j} \nabla x_j$$

ขั้นตอนวิธี forward mode

กำหนดให้ $f : \mathbf{R}^n \to \mathbf{R}$ ต้องการหาค่าเกรเดียนต์ของ $f(x_1, x_2, ..., x_n)$

ขั้นเริ่มต้น

กำหนดให้ $i = n+1$

ขั้นหลัก

1. สมมติให้ x_i เป็น ตัวแปรระหว่างกลาง (intermediate variable)
2. แยก $f(x_1, x_2, ..., x_n)$ ออกเป็นฟังก์ชันมูลฐาน โดยเลือกฟังก์ชันมูลฐานที่อยู่ในฟังก์ชัน f หรือผลการบวก ลบ คูณ หาร ของตัวแปรระหว่างกลางที่กำหนดไว้ก่อนหน้านี้

   กำหนดให้ $x_i = f_i(x_k, k \in J_i)$

   เมื่อ J_i คือเซตดรรชนีที่บอกว่า x_k เป็นตัวแปรอิสระของฟังก์ชันมูลฐาน f_i

   โดยกฎลูกโซ่ $\nabla x_i = \sum_{j \in J_i} \frac{\partial f_i(x_k, k \in J_i)}{\partial x_j} \nabla x_j$
3. ตรวจสอบว่า $x_i = f_i(x_k, k \in J_i) = f(x_1, x_2, ..., x_n)$ หรือไม่
4. ถ้า $x_i \neq f(x_1, x_2, ..., x_n)$ แล้ว กำหนดให้ $i = i+1$ และ กลับไปขั้นตอนที่ 3

   ถ้า $x_i = f(x_1, x_2, ..., x_n)$ แล้ว กำหนดให้ $m = i$ จะได้ว่า $\nabla f = \nabla x_m$ แล้วหยุดการคำนวณ

ตัวอย่าง การคำนวณหาค่าเกรเดียนต์ของฟังก์ชันโดยวิธี forward mode

กำหนดให้ $f(x_1, x_2) = (x_1 + x_2)x_1 + sin(x_2)$

จะเห็นได้ว่า f ไม่เป็นฟังก์ชันมูลฐาน จะทำการแยก f ออกเป็นฟังก์ชันมูลฐานดังนี้

กำหนดตัวแปรระหว่างกลาง $x_3 = f_3(x_1, x_2) = x_1 + x_2$ เพราะฉะนั้น $J_3 = \{1,2\}$

โดยกฎลูกโซ่ จะได้ว่า $\nabla x_3 = \frac{\partial f_3}{\partial x_1}\nabla x_1 + \frac{\partial f_3}{\partial x_2}\nabla x_2 = \nabla x_1 + \nabla x_2$

กำหนดตัวแปรระหว่างกลาง $x_4 = f_4(x_3, x_1) = x_3 x_1$ เพราะฉะนั้น $J_4 = \{1,3\}$

โดยกฎลูกโซ่ จะได้ว่า $\nabla x_4 = \frac{\partial f_4}{\partial x_3}\nabla x_3 + \frac{\partial f_4}{\partial x_1}\nabla x_1 = x_1 \nabla x_3 + x_3 \nabla x_1$

กำหนดตัวแปรระหว่างกลาง $x_5 = f_5(x_2) = sin(x_2)$ เพราะฉะนั้น $J_5 = \{2\}$

โดยกฎลูกโซ่ จะได้ว่า $\nabla x_5 = \frac{\partial f_5}{\partial x_2}\nabla x_2 = cos(x_2)\nabla x_2$

กำหนดตัวแปรระหว่างกลาง $x_6 = f_6(x_4, x_5) = x_4 + x_5$ เพราะฉะนั้น $J_6 = \{4,5\}$

โดยกฎลูกโซ่ จะได้ว่า $\nabla x_6 = \frac{\partial f_6}{\partial x_4}\nabla x_4 + \frac{\partial f_6}{\partial x_5}\nabla x_5 = \nabla x_4 + \nabla x_5$

จะเห็นได้ว่า $x_6 = f_6(x_4, x_5) = x_4 + x_5 = f(x_1, x_2) = (x_1 + x_2)x_1 + sin(x_2)$

เพราะฉะนั้น $\nabla f(x_1, x_2) = \nabla x_6 = \nabla x_4 + \nabla x_5$

$$\begin{aligned}
&= x_1 \nabla x_3 + x_3 \nabla x_1 + cos(x_2)\nabla x_2 \\
&= x_1(\nabla x_1 + \nabla x_2) + x_3 \nabla x_1 + cos(x_2)\nabla x_2 \\
&= (x_1 + x_3)\nabla x_1 + (x_1 + cos(x_2))\nabla x_2 \\
&= (2x_1 + x_2)e_1 + (x_1 + cos(x_2))e_2 \\
&= (2x_1 + x_2, x_1 + cos(x_2))
\end{aligned}$$

3.1.2 วิธี reverse mode

ในการหาค่าเกรเดียนต์โดยวิธี reverse mode ต้องการหาค่าเกรเดียนต์ของ $f(x_1, x_2, ..., x_n)$ ซึ่งเป็นฟังก์ชันของตัวแปรอิสระ n ตัวและฟังก์ชันที่ซับซ้อนที่ไม่เป็นฟังก์ชันมูลฐาน เริ่มต้นด้วยการแยกฟังก์ชันที่ซับซ้อนที่ต้องการหาค่าเกรเดียนต์ออกเป็นฟังก์ชันมูลฐาน กำหนดตัวแปรระหว่างกลางให้มีค่าเท่ากับฟังก์ชันมูลฐานที่แยกออกมาหรือผลบวก ลบ คูณ หารของตัวแปรระหว่างกลางที่กำหนดไว้ก่อนหน้า กระทำการแยกไปเรื่อยๆ จนกระทั่งตัวแปรระหว่างกลางสุดท้ายคือฟังก์ชันที่ต้องการหาเกรเดียนต์สามารถอธิบายในรูปของสัญลักษณ์ได้ดังนี้

กำหนดให้ $i = n + 1$

กำหนดตัวแปรระหว่างกลาง (intermediate variable) x_i ให้มีเท่ากับฟังก์ชันมูลฐานที่แยกออกมา นั่นคือ

$$x_i = f_i(x_k, k \in J_i)$$

เมื่อ J_i คือเซตดรรชนีที่บอกว่า x_k เป็นตัวแปรอิสระของฟังก์ชันมูลฐาน f_i

กำหนดให้ $i = n + 2$

กำหนดตัวแปรระหว่างกลาง x_i ให้มีค่าเท่ากับฟังก์ชันมูลฐาน นั่นคือ $x_i = f_i(x_k, k \in J_i)$

เพิ่มค่า i กระทำในลักษณะแยกฟังก์ชันมูลฐานและการบวก ลบ คูณ หาร ของตัวแปรระหว่างกลาง ไปเรื่อยๆ จนกระทั่ง ตัวแปรระหว่างกลางสุดท้ายคือฟังก์ชันที่ต้องการหาค่าเกรเดียนต์ f

สมมติว่า $i = m$ ได้ $\quad x_i = f_i(x_k, k \in J_i) = f(x_1, x_2, ..., x_n)$

นั่นคือ $\quad x_m = f_m(x_k, k \in J_m) = f(x_1, x_2, ..., x_n)$

เมื่อ m คือจำนวนเต็มบวกที่ทำให้ตัวแปรระหว่างกลาง $x_m = f(x_1, x_2, ..., x_n)$

เมื่อแยกออกเป็นฟังก์ชันมูลฐานทั้งหมดแล้ว ขั้นตอนหาค่าเกรเดียนต์โดยการกำหนด ตัวแปรผูกพัน (adjoint variable) $y_i \ , i = 1,2,3,....,m$ คือค่าอนุพันธ์ย่อยของ $f(x_1, x_2, ..., x_n)$ เทียบกับ x_i

นั่นคือ $y_i = \dfrac{\partial f}{\partial x_i} \ , \ i = 1,2,3,....,m$

จะได้ว่า $y_m = \dfrac{\partial f}{\partial x_m} = \dfrac{\partial x_m}{\partial x_m} = 1$

เพราะฉะนั้น เริ่มต้น กำหนดให้ $y_m = 1$ และ $y_i = 0$ สำหรับ $i = 1,2,3,....,m-1$

คำนวณหาค่า $y_i \ , i = 1,2,3,....,m$ โดยใช้กฎลูกโซ่

เริ่มที่ $i = m$ จะได้ $y_j = y_j + \left[\dfrac{\partial f_m(x_k, k \in J_m)}{\partial x_j}\right] y_m$ สำหรับทุก $j \in J_m$

$i = m-1$ จะได้ $y_j = y_j + \left[\dfrac{\partial f_i(x_k, k \in J_{m-1})}{\partial x_j}\right] y_{m-1}$ สำหรับทุก $j \in J_{m-1}$

....

$i = n+1$ จะได้ $y_j = y_j + \left[\dfrac{\partial f_i(x_k, k \in J_{n+1})}{\partial x_j}\right] y_{n+1}$ สำหรับทุก $j \in J_{n+1}$

แล้วสรุปผลรวมค่าของ $y_i \ , i = 1,2,3,....,m$

ผลลัพธ์สุดท้าย จะได้ค่าเกรเดียนต์ของฟังก์ชัน $f(x_1, x_2, ..., x_n)$ คือ

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n}\right) = (y_1, y_2, ..., y_n)$$

ขั้นตอนวิธี reverse mode

กำหนดให้ $f : \mathbf{R}^n \to \mathbf{R}$ ต้องการหาค่าเกรเดียนต์ของ $f(x_1, x_2, ..., x_n)$

ขั้นเริ่มต้น

กำหนดให้ $i = n+1$

ขั้นหลัก

1. กำหนดให้ x_i เป็น ตัวแปรระหว่างกลาง (intermediate variable)
2. แยก $f(x_1, x_2, ..., x_n)$ ออกเป็นฟังก์ชันมูลฐาน โดยเลือกฟังก์ชันมูลฐานที่อยู่ในฟังก์ชัน f หรือผลการบวก ลบ คูณ หาร ของตัวแปรระหว่างกลางที่กำหนดไว้ก่อนหน้านี้

   กำหนดให้ $x_i = f_i(x_k, k \in J_i)$

   เมื่อ J_i คือเซตดรรชนีที่บอกว่า x_k เป็นตัวแปรอิสระของฟังก์ชันมูลฐาน f_i

3. ตรวจสอบว่า $x_i = f_i(x_k, k \in J_i) = f(x_1, x_2, ..., x_n)$ หรือไม่

   ถ้าเป็นจริง แล้ว กำหนดให้ $m = i$ กระทำที่ขั้นตอนที่ 4

   ถ้าเป็นเท็จ แล้ว กำหนดให้ $i = i+1$ และ กลับไปขั้นตอนที่ 3

4. กำหนดตัวแปรผูกพัน y_i , $i = 1,2,3,....,m$

   และกำหนดค่า $y_i = 0$ สำหรับ $i = 1,2,3,....,m-1$ และ $y_m = 1$

5. สำหรับค่า $i = m$ ลดลงจนถึง $n+1$

$$y_j = y_j + \left[\frac{\partial f_i(x_k, k \in J_i)}{\partial x_j}\right] y_i \quad \text{สำหรับทุก } j \in J_i$$

6. จะได้ $\nabla f = (y_1, y_2, ..., y_n)$

ตัวอย่าง การคำนวณหาค่าเกรเดียนต์ของฟังก์ชันโดยวิธี reverse mode

กำหนดให้ $f(x_1, x_2) = (x_1 + x_2)x_1 + sin(x_2)$

จะเห็นได้ว่า f ไม่เป็นฟังก์ชันมูลฐาน จะทำการแยก f ออกเป็นฟังก์ชันมูลฐานดังนี้

กำหนดตัวแปรระหว่างกลาง $x_3 = f_3(x_1, x_2) = x_1 + x_2$ เพราะฉะนั้น $J_3 = \{1,2\}$

$x_4 = f_4(x_3, x_1) = x_3 x_1$ เพราะฉะนั้น $J_4 = \{1,3\}$

$x_5 = f_5(x_2) = sin(x_2)$ เพราะฉะนั้น $J_5 = \{2\}$

$x_6 = f_6(x_4, x_5) = x_4 + x_5$ เพราะฉะนั้น $J_6 = \{4,5\}$

จะเห็นได้ว่า $x_6 = f_6(x_4, x_5) = x_4 + x_5 = f(x_1, x_2) = (x_1 + x_2)x_1 + sin(x_2)$

กำหนด ตัวแปรผูกพัน (adjoint variable)

y_i , $i = 1,2,3,4,5,6$ คือค่าอนุพันธ์ย่อยของ $f(x_1, x_2, ..., x_n)$ เทียบกับ x_i

นั่นคือ $y_i = \dfrac{\partial f}{\partial x_i}$, $i = 1,2,3,4,5,6$

เริ่มต้นกำหนดให้ $y_1 = y_2 = y_3 = y_4 = y_5 = 0$ และ $y_6 = 1$

คำนวณหาค่า y_i , $i = 1,2,3,4,5,6$ โดยใช้กฎลูกโซ่จากฟังก์ชันมูลฐานที่แยกออกมา

จาก $x_6 = f_6(x_4, x_5) = x_4 + x_5$, $J_6 = \{4,5\}$ จะได้ $y_4 = y_4 + \left(\dfrac{\partial f_6}{\partial x_4}\right) y_6 = 1$

$$y_5 = y_5 + \left(\frac{\partial f_6}{\partial x_5}\right) y_6 = 1$$

จาก $x_5 = f_5(x_2) = sin(x_2)$, $J_5 = \{2\}$ จะได้ $y_2 = y_2 + \left(\dfrac{\partial f_5}{\partial x_2}\right) y_5 = cos(x_2)$

จาก $x_4 = f_4(x_3, x_1) = x_3 x_1$, $J_4 = \{1,3\}$ จะได้ $y_3 = y_3 + \left(\frac{\partial f_4}{\partial x_3}\right) y_4 = x_1$

$$y_1 = y_1 + \left(\frac{\partial f_4}{\partial x_1}\right) y_4 = x_3$$

จาก $x_3 = f_3(x_1, x_2) = x_1 + x_2$, $J_3 = \{1,2\}$ จะได้ $y_1 = y_1 + \left(\frac{\partial f_3}{\partial x_1}\right) y_3 = x_3 + x_1$

$$y_2 = y_2 + \left(\frac{\partial f_3}{\partial x_2}\right) y_3 = cos(x_2) + x_1$$

จะได้ค่าเกรเดียนต์ของฟังก์ชัน $f(x_1, x_2)$ คือ

$$\begin{aligned}\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}\right) &= (y_1, y_2) \\ &= (x_3 + x_1, x_1 + cos(x_2)) \\ &= (2x_1 + x_2, x_1 + cos(x_2))\end{aligned}$$

3.2 โปรแกรมคำนวณหาค่าเกรเดียนต์ของฟังก์ชัน ADOL-C

ในการสร้างซอฟต์แวร์หาผลเฉลยของปัญหากำหนดการไม่เชิงเส้นนี้มีขั้นตอนเรียกใช้โปรแกรม A Package for the Automatic Differentiation of Algorithms Written in C/C++ (ADOL-C) [6,8] คำนวณหาค่าเกรเดียนต์ของฟังก์ชันจุดประสงค์เพื่อกำหนดให้เป็นสัมประสิทธิ์ของฟังก์ชันจุดประสงค์ของปัญหากำหนดการเชิงเส้นใช้สำหรับคำนวณหาทิศทางที่เป็นไปได้ ADOL-C พัฒนาขึ้นโดยคณะนักวิทยาศาสตร์ของสถาบัน Institute of Scientific Computing ,Technical University Dresden , Germany ในปี ค.ศ. 1999 เขียนโดยใช้ภาษา C/C++ บนระบบปฏิบัติการ Linux มีการสร้าง header file และlibrary file ขึ้นมาใช้งานเอง ADOL-C คำนวณหาค่าอนุพันธ์ของฟังก์ชันโดยใช้วิธีการของ Automatic differentiation ทั้งรูปแบบ forward mode และ reverse mode การใช้งานโปรแกรม ADOL-C ต้องเขียนเป็นโปรแกรมภาษา C/C++ ให้ถูกต้องกับรูปแบบกับไวยากรณ์ของ ADOL-C ที่กำหนดไว้ซึ่งมีความแตกต่างจากโปรแกรมภาษา C/C++ เช่น การประกาศตัวแปร ฟังก์ชันการใช้งานเฉพาะของการหาค่าเกรเดียนต์ จาโคเบียน Hessian รวมทั้งฟังก์ชันการหาอนุพันธ์อันดับสูง

ตัวอย่าง Drivers ของ ADOL-C สำหรับงานด้าน Optimization การใช้งานฟังก์ชันในขั้นตอนการแปลภาษาโปรแกรมให้เชื่อมกับ libad.a ซึ่งเป็น library ที่ได้จากการติดตั้งโปรแกรม ADOL-C

```
int gradient(tag,n,x,g)
short int tag;        // tape identification
int n;                // number of independent variables n
double x[n];          // independent vector x
double g[n];          // resulting gradient f(x)

int jacobian(tag,m,n,x,J)
short int tag;        // tape identification
int m;                // number of dependent variables m
int n;                // number of independent variables n
double x[n];          // independent vector x
double J[m][n];       // resulting Jacobian f(x)

int hessian(tag,n,x,H)
short int tag;        // tape identification
int n;                // number of independent variables n
double x[n];          // independent vector x
double H[n][n];       // resulting Hessian matrix f(x)
```

ตัวอย่าง การเขียนโปรแกรมคำนวณหาเกรเดียนต์ของฟังก์ชันของ ADOL-C

หาเกรเดียนต์ของ $f(x_1, x_2) = (x_1 + x_2)x_1 + sin(x_2)$

สามารถเขียนเป็นโปรแกรมภาษา C++ และเรียกใช้ฟังก์ชัน gradient ของ ADOL-C ได้ดังนี้

```
#include "adolc.h"                  // ประกาศ header file ของ ADOL-C
#include <iostream.h>               // ประกาศ header file ของภาษา C++
using namespace std;
int main(void)
{
  int i  , n ;
  int tag=1;
  double xp[n];
  double yp;                        ประกาศตัวแปร
  double grad[n];
  adouble *x = new adouble[n];
  adouble y;
  printf("\nnumber of independent variable = %d\n\n",n);
 for(i=0; i<n; i++)
  xp[i] = 0;                        // กำหนดค่าของจุด x ที่ต้องการหา ∇f (x)
```

```
trace_on(tag);
  for(i=0;i<n;i++)
        x[i] <<=xp[i];                    //กำหนดค่าให้ตัวแปรอิสระ

  y = (x[0]+x[1])*x[0]+sin(x[1]);       //กำหนดฟังก์ชัน
  y >>= yp;                             // กำหนดค่าให้ตัวแปรตาม
  delete[] x;
trace_off(tag);

gradient(tag,n,xp,grad);                //ฟังก์ชันหาค่าเกรเดียนต์ของฟังก์ชัน
for(i=0;i<n;i++)
     printf("%lf\t",grad[I])            //แสดงผล ∇f (x)
return 0;
}
```

จากตัวอย่างของโปรแกรม ADOL-C จะเห็นว่ามีการประกาศตัวแปร Active Variables เป็น adouble และการหาค่าเกรเดียนต์เรียกใช้ฟังก์ชัน gradient(tag,n,xp,grad) ซึ่งไม่ใช่ฟังก์ชันของภาษา C/C++ เป็นฟังก์ชันที่สร้างขึ้นมาโดยโปรแกรม ADOL-C

3.3 โปรแกรมคำนวณหาผลเฉลยกำหนดการเชิงเส้น GNU Linear Programming Kit (GLPK)

GLPK [9] เป็นโปรแกรมสำหรับหาผลเฉลยของปัญหากำหนดการเชิงเส้นโดยวิธี Revised simplex method พัฒนาขึ้นในปี ค.ศ. 2000 โดยนักวิทยาการคอมพิวเตอร์ชาวรัสเซีย Andrew Makhorin , Department for Applied Informatics, Moscow Aviation Institute, Moscow, Russia. โปรแกรม GLPK สร้าง header file มีลักษณะเหมือน header file ของภาษา C และสร้าง library file คือ libglpk.a ขึ้นมาใช้งานเอง รูปแบบการการเรียกใช้โปรแกรมจะสามารถกำหนดคำสั่งของการแก้ปัญหาในรูปแบบภาษา C และฟังก์ชันของ GLPK ในการสร้างซอฟต์แวร์หาผลเฉลยปัญหากำหนดการไม่เชิงเส้น เรียกใช้โปรแกรม GLPK แก้ปัญหากำหนดการเชิงเส้นคำนวณหาทิศทางที่เป็นไปได้ในการเพิ่มค่าฟังก์ชันจุดประสงค์

ตัวอย่าง การใช้งานโปรแกรม GLPK หาผลเฉลยกำหนดการเชิงเส้น

การหาค่าสูงสุดของฟังก์ชันจุดประสงค์ $z = 6d_1 + 5d_2$

ภายใต้เงื่อนไข

$$d_1 + d_2 \le 2$$
$$d_1 + 2d_2 \le 3$$
$$d_1 \ge 0 \;,\; d_2 \ge 0$$

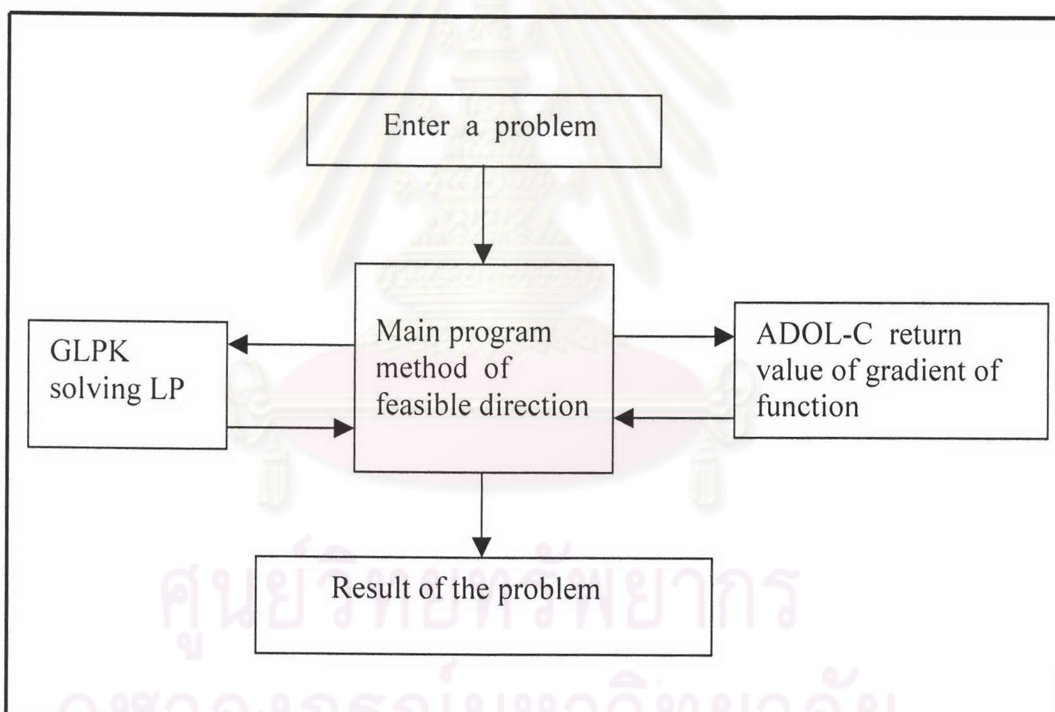
จากปัญหาสามารถเขียนโปรแกรมภาษา C รวมทั้งคำสั่งของ GLPK ได้ดังนี้

```
#include <stdio.h>
#include <stdlib.h>
#include "glpk.h"

int main(void)
{    LPX *lp;
     int rn[1+4], cn[1+4];
     double a[1+4], Z, d1, d2 ;

lp = lpx_create_prob();
lpx_set_prob_name(lp, "sample");
lpx_add_rows(lp, 2);
lpx_set_row_name(lp, 1, "p");
lpx_set_row_bnds(lp, 1, LPX_UP, 0.0, 2.0);
lpx_set_row_name(lp, 2, "q");
lpx_set_row_bnds(lp, 2, LPX_UP, 0.0, 3.0);
lpx_add_cols(lp, 2);
lpx_set_col_name(lp, 1, "d1");
lpx_set_col_bnds(lp, 1, LPX_LO, 0.0, 0.0);
lpx_set_col_name(lp, 2, "d2");
lpx_set_col_bnds(lp, 2, LPX_LO, 0.0, 0.0);
rn[1] = 1, cn[1] = 1, a[1] =  1.0;
rn[2] = 1, cn[2] = 2, a[2] =  1.0;
rn[4] = 2, cn[4] = 1, a[4] = 1.0;
rn[6] = 2, cn[6] = 2, a[6] =  2.0;
lpx_load_mat3(lp, 4, rn, cn, a);
lpx_set_obj_dir(lp, LPX_MAX);
lpx_set_col_coef(lp, 1, 6.0);
lpx_set_col_coef(lp, 2, 5.0);
lpx_simplex(lp);
 Z = lpx_get_obj_val(lp);
 lpx_get_col_info(lp, 1, NULL, &d1, NULL);
 lpx_get_col_info(lp, 2, NULL, &d2, NULL);

 printf("\nZ = %g; x1 = %g; x2 = %g\n", Z, d1, d2);
 lpx_delete_prob(lp);
 return 0;
}
```

เมื่อคำนวณแล้วได้ผลเฉลยเป็น $\mathbf{d} = (d_1, d_2) = (2.00, 0.00)$

3.4 การออกแบบซอฟต์แวร์

การสร้างซอฟต์แวร์พัฒนาขึ้นโดยภาษา C และC ++ บนระบบปฏิบัติการ Linux Mandrake Version 9.0 ตั้งชื่อซอฟต์แวร์ว่า MFD เริ่มต้นเขียนโปรแกรมหลักตามขั้นตอนการหาผลเฉลยของวิธีการใช้ทิศทางที่เป็นไปได้ แล้วเขียนโปรแกรม ADOL-C คำนวณค่าเกรเดียนต์แล้วส่งค่าให้โปรแกรมหลักและเขียนโปรแกรม GLPK รับข้อมูลจากโปรแกรมหลักแล้วคำนวณหาผลเฉลยกำหนดการเชิงเส้นได้ผลลัพธ์ส่งค่ากลับมาที่โปรแกรมหลัก โปรแกรมหลักจะควบคุมการทำงานตามวิธีการใช้ทิศทางที่เป็นไปได้ จนกระทั่งแสดงผลลัพธ์ออกมา สามารถเขียนในรูปของ software design ได้ดังนี้



3.5 ขั้นตอนวิธีของซอฟต์แวร์ที่ต้องการ

การออกแบบซอฟต์แวร์สำหรับหาผลเฉลยของปัญหากำหนดการไม่เชิงเส้นภายใต้เงื่อนไขบังคับเชิงเส้น ได้สร้างขั้นตอนวิธีเพื่อเป็นเครื่องมือในการเขียนโปรแกรมภาษา C ดังนี้

การหาค่าสูงสุดของฟังก์ชันจุดประสงค์ $z = f(\mathbf{x})$

ภายใต้เงื่อนไข $\mathbf{Ax} \le \mathbf{b}$

$\mathbf{x} \ge \mathbf{0}$

ขั้นเริ่มต้น

กำหนดให้ EPS คือความยาวของระยะห่างจุดถัดไปที่ยอมรับได้

กำหนดให้ $k = 1$

เลือกจุดเริ่มต้น $\mathbf{x}^1$ เป็นจุดที่อยู่ในบริเวณที่เป็นไปได้ แล้วเข้าสู่ขั้นหลัก

ขั้นหลัก

1. คำนวณหาค่าเกรเดียนต์ $\nabla f\left(\mathbf{x}^k\right)$ โดยวิธี Automatic differentiation เป็นการเรียกใช้โปรแกรม ADOL-C
2. คำนวณหา $\mathbf{d}^k$ ซึ่งเป็นการหาผลเฉลยของปัญหากำหนดการเชิงเส้น

   การหาค่าสูงสุดของฟังก์ชันจุดประสงค์ $z = \nabla f\left(\mathbf{x}^k\right)^T \mathbf{d}$

   ภายใต้เงื่อนไข $\mathbf{Ad} \le \mathbf{b}$

   $\mathbf{d} \ge \mathbf{0}$

   ในขั้นตอนนี้เรียกใช้โปรแกรม GLPK
3. คำนวณหาระยะเคลื่อนที่ α_k จากปัญหากำหนดการไม่เชิงเส้น

   การหาค่าสูงสุดของฟังก์ชันจุดประสงค์ $z = f\left(\mathbf{x}^k + \alpha_k\left(\mathbf{d}^k - \mathbf{x}^k\right)\right)$

   ภายใต้เงื่อนไข $0 \le \alpha_k \le 1$

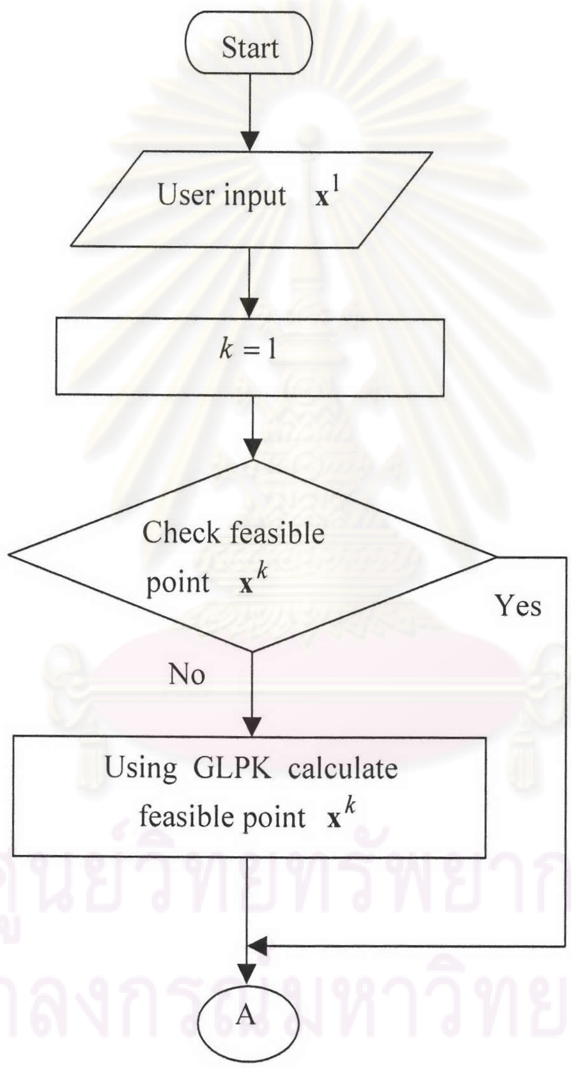
   ในขั้นตอนนี้เรียกใช้ส่วนของโปรแกรมที่คำนวณวิธี golden section หรือกฎของ Armijo หรือ วิธี bisection
4. กำหนดให้จุด $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k\left(\mathbf{d}^k - \mathbf{x}^k\right)$
5. ตรวจสอบค่า $\left\|\mathbf{x}^{k+1} - \mathbf{x}^k\right\| \le$ EPS หรือไม่

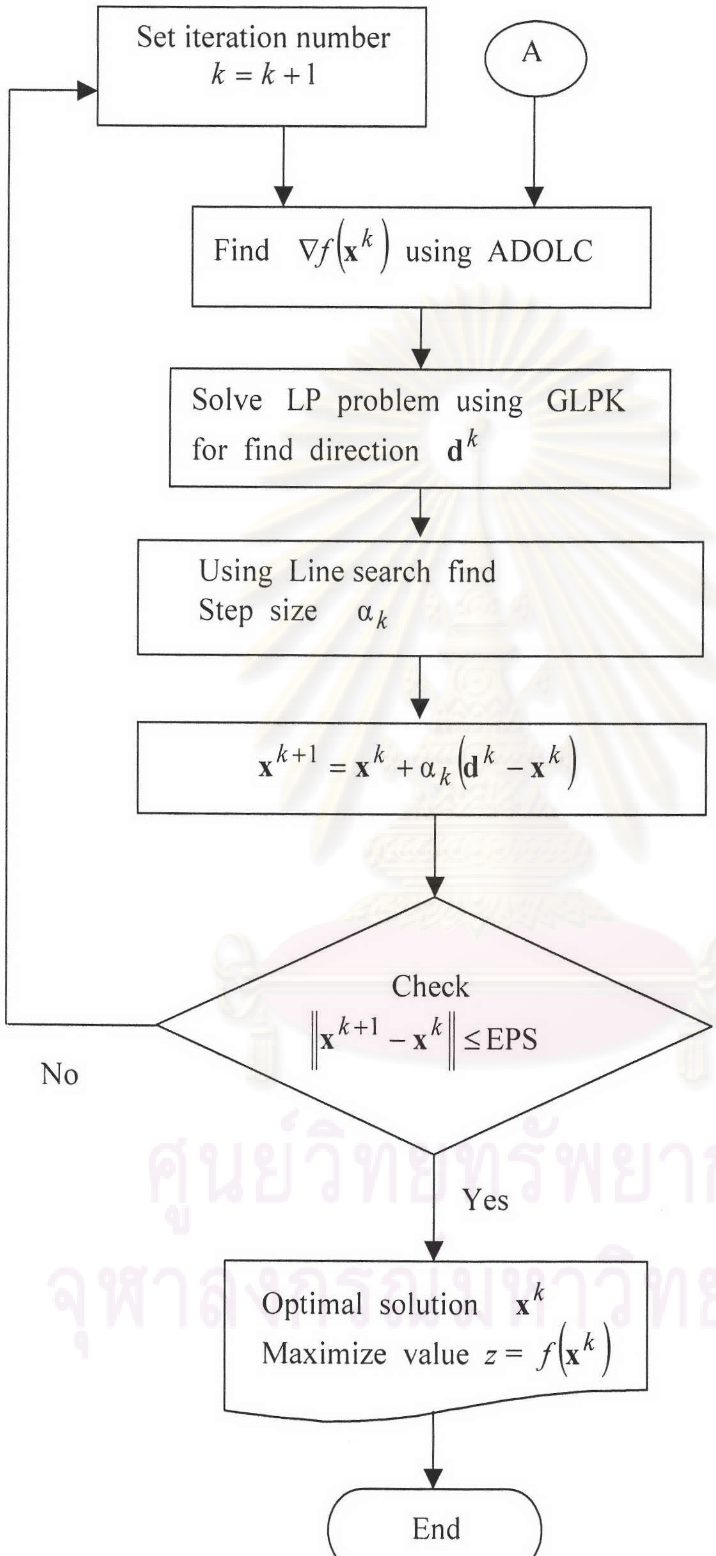
   ถ้าเป็นจริงแล้วหยุดการคำนวณสรุปว่า $\mathbf{x}^k$ เป็นผลเฉลยเหมาะที่สุดของปัญหา

   ถ้าเป็นเท็จให้ทำขั้นตอนที่ 6
6. กำหนดให้ $k = k + 1$ แล้วกลับไปทำขั้นตอนที่ 1.

3.6 ผังงาน

การออกแบบซอฟต์แวร์สำหรับหาผลเฉลยของปัญหากำหนดการไม่เชิงเส้นภายใต้เงื่อนไขบังคับเชิงเส้นได้สร้างผังงานเพื่อเป็นเครื่องมือในการเขียนโปรแกรมภาษา C ดังนี้




Set iteration number
$k = k + 1$
A
Find $\nabla f(\mathbf{x}^k)$ using ADOLC
Solve LP problem using GLPK
for find direction $\mathbf{d}^k$
Using Line search find
Step size α_k
$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k(\mathbf{d}^k - \mathbf{x}^k)$
Check
$\|\mathbf{x}^{k+1} - \mathbf{x}^k\| \le \mathrm{EPS}$
No
Yes
Optimal solution $\mathbf{x}^k$
Maximize value $z = f(\mathbf{x}^k)$
End

3.7 วิธีการใช้งานซอฟต์แวร์ที่สร้างขึ้น

ซอฟต์แวร์ MFD ที่สร้างขึ้น บีบอัดไฟล์ข้อมูลทั้งหมดอยู่ในไฟล์ MFD.tar.gz เริ่มต้นการใช้งานต้องขยายไฟล์ออกมาโดยใช้คำสั่ง

> gunzip MFD.tar.gz และ
> tar –xvf MFD.tar

เมื่อขยายออกมาแล้วได้ผลลัพธ์เป็นไดเรกทอรี MFD ซึ่งประกอบด้วยไฟล์และกลุ่มไฟล์ที่อยู่ในไดเรกทอรีดังนี้

1. Directory ประกอบด้วย 3 กลุ่มคือ

include เป็นไดเรกทอรีที่ประกอบไปด้วย header file ของ GLPK

SRC เป็นไดเรกทอรีที่ประกอบไปด้วย header file ของ ADOL-C

LIB เป็นไดเรกทอรีที่ประกอบไปด้วย library file ของ GLPK คือ libglpk.a และ library file ของ ADOL-C คือ libad.a

2. File ประกอบด้วย 6 ไฟล์คือ

adolc.c เป็นไฟล์ที่ใช้ในการคำนวณหาค่าเกรเดียนต์ของฟังก์ชันจุดประสงค์

fea.c เป็นไฟล์หลักควบคุมการทำงานตามวิธีการใช้ทิศทางที่เป็นไปได้

adfunc.c เป็นไฟล์ที่เตรียมสมการฟังก์ชันจุดประสงค์ให้กับไฟล์ adolc.c

objective.c เป็นไฟล์ที่เตรียมสมการฟังก์ชันจุดประสงค์ให้กับไฟล์ fea.c

constraint.txt เป็นไฟล์ข้อมูลของสมการเงื่อนไขบังคับ

makefile เป็นไฟล์ที่สั่งให้คอมพิวเตอร์แปลโปรแกรมเป็นภาษาเครื่องคอมพิวเตอร์

เริ่มต้นการใช้งาน ผู้ใช้งานต้องเตรียมไฟล์ข้อมูล objective.c, constraint.txt, adfunc.c ตามรูปแบบที่กำหนดสำหรับการใช้งานซอฟต์แวร์ พิมพ์คำสั่ง > makefile คอมพิวเตอร์แปลโปรแกรมเป็นภาษาเครื่อง ผลลัพธ์ได้ adolc.o , fea.o, adfunc.o, objective.o และคอมพิวเตอร์รวมไฟล์ adfunc.o และ objective.o เป็น library file คือ libfea.a พร้อมทั้งจะมีไฟล์ที่แสดงผลลัพธ์ด้วยคือ output สามารถดูผลลัพธ์ที่ได้จากการพิมพ์คำสั่ง >./output คอมพิวเตอร์จะแสดงผลการคำนวณออกมา สามารถดูสถิติการคำนวณทั้งหมดจากไฟล์ output.txt และสามารถนำผลการคำนวณจากไฟล์ graph.txt ไปแสดงผลในรูปของกราฟได้จากโปรแกรม gnuplot หรือ Microsoft Excel

ตัวอย่าง การเตรียมไฟล์ข้อมูลเพื่อใช้งานของซอฟต์แวร์

การหาค่าต่ำสุดของฟังก์ชันจุดประสงค์ $z = f(\mathbf{x}) = x_1^3 + 2x_2^3 - x_1^2 + x_1 - 2x_2$

ภายใต้เงื่อนไข

$$x_1 + 2x_2 \le 6$$
$$-x_1 + 2x_2 \le 3$$
$$x_1 \ge 0,\ x_2 \ge 0$$

ไฟล์ **objective.c** เป็นไฟล์ของฟังก์ชันจุดประสงค์ เตรียมไฟล์ในรูปแบบดังนี้

```
double func(double x[])
        {
         double f
                 f = pow(x[0],3)+2*pow(x[1],3)-pow(x[0],2)+x[0]-2*x[1];
         return f;
        }
```

ไฟล์ **adfunc.c** เตรียมข้อมูลเพื่อให้ ADOL-C คำนวณหาค่าเกรเดียนต์ของฟังก์ชันจุดประสงค์

```
adouble func(adouble x[])
        {
         adouble f
                 f = pow(x[0],3)+2*pow(x[1],3)-pow(x[0],2)+x[0]-2*x[1];
         return f;
        }
```

หมายเหตุ ข้อมูลไฟล์ **adfunc.c** แตกต่างจากไฟล์ **objective.c** ที่การประกาศตัวแปรจาก double เป็น adouble เพราะว่าเป็นรูปแบบการประกาศตัวแปรของ ADOL-C

ไฟล์ **constraint.txt** ข้อมูลเกี่ยวกับเงื่อนไขบังคับและข้อมูลที่เป็นเงื่อนไขของการหาผลเฉลย

2			// 1: maximize, 2:minimize
2			// number of independent variable
2			// จำนวนแถวของเงื่อนไขบังคับ
2			// จำนวนหลักของเงื่อนไขบังคับ
1	2	6	ค่าสัมประสิทธิ์ของเงื่อนไขบังคับ **Ax ≤ b**
-1	2	3	
0	0		// initial point $\mathbf{x}^1$
0	0.58		// exact Solution
100000			// maximum iteration
0.001			// EPS
1			// 1:automatic differentiation , 2:central difference
0.0001			// ค่า h ในกรณีใช้ Central Difference
2			// 1:golden section, 2: Armijo's rule, 3 : bisection

ตัวอย่าง ไฟล์ผลลัพธ์ของการประมวลผลซอฟต์แวร์

ไฟล์ **output** เมื่อพิมพ์คำสั่ง >./output จะแสดงผลลัพธ์ออกมาดังนี้

```
Number of Iteration = 4
Optimal value z  = -0.769800 at point :(0.000000,0.577475)
Exact solution = (0.00,0.58)
Relative error =0.435421 %
System time = 0.02
```

ไฟล์ **output.txt**

```
Type of the problem is a Minimize:
Calculate gradient using Automatic Differentiation
Line search using  Armijo's Rule

                Summary of Computations
-----------------------------------------------------------------------------------------------------
k    x(k)         f(x(k))  gf(x(k))       d(k)         step_size  x(k+1)       residual

1  (0.00,0.00)  0.00   (1.00,-2.00)  (0.00,1.50)  0.200000  (0.00,0.30)  0.300000
2  (0.00,0.30)  -0.55   (1.00,-1.46)  (0.00,1.50)  0.200000  (0.00,0.54)  0.240000
3  (0.00,0.54)  -0.77   (1.00,-0.25)  (0.00,1.50)  0.040000  (0.00,0.58)  0.038400
4  (0.00,0.58)  -0.77   (1.00,0.01)   (0.00,0.00)  0.001600  (0.00,0.58)  0.000925
-----------------------------------------------------------------------------------------------------

OPTIMAL VALUE z = -0.769800 at point :(0.000000,0.577475)
 relative error = 0.435421 %
 accuracy = 99.564579 %
```

ไฟล์ **graph.txt** เป็นข้อมูลที่สามารถนำไปวาดกราฟได้

```
0.000000      0.000000
0.000000      0.300000
0.000000      0.540000
0.000000      0.578400
```

เมื่อนำข้อมูลจากไฟล์ graph.txt มาให้โปรแกรม Microsoft Excel วาดกราฟได้ผลดังรูป