Revista de Matemática: Teoría y Aplicaciones 1999 **6**(2) : 153–174
cimpa – ucr – ccss issn: 1409-2433

OPTIMIZA – UN PAQUETE COMPUTACIONAL PARA LA OPTIMIZACIÓN DE PROBLEMAS NO LINEALES

Gonzalo Palencia F.*– Vicente Molina P.* – Venzyslav Llano R.*

Recibido: 5 agosto 1998

Resumen

El objetivo del trabajo es construir un software capaz de solucionar problemas de optimización no lineal. Se centra una especial atención en la selección del algoritmo que utiliza basado en el método de Lagrangiano Aumentado combinado con métodos quasi-Newton ($BFGS, L-BFGS$). Se explica como está estructurado internamente el software, los pasos para su construcción, la forma de usarlo y se analizan los resultados de las pruebas numéricas realizadas. El sotware OPTIMIZA 3.0 se ejecuta sobre una plataforma Windows (realizado en Borland Delphi 3.0), sus capacidades en cuanto a cantidad de variables y restricciones solo están limitadas por la capacidad de memoria de la máquina utilizada.

Palabras Clave: Optimización no lineal, optimización con restricciones, programación matemática, programación no lineal, optimización.

Abstract

Our goal is to build a software able to solve problems in non linear optimization. A central point is selection of the algorithm, which is based on Augmented Lagrangian Method combined with quasi-Newton methods (BFGS, L-BFGS). The article explains how software is structured, the steps of its construction and the mode of use; furthermore, results and numerical tests are analyzed. OPTIMIZA 3.0 was built on Borland Delphi 3.0 and runs on Windows, its capacity on the number of variables and constraints are only limited on the memory machine to be used.

Keywords: Nonlinear optimization, constrained optimization, mathematical programming, nonlinear programming, optimization.

AMS Subject Classification: 90C30, 68-04

* Universidad Central "Marta Abreu" de Las Villas, Facultad de Matemática, Física y Computación, Departamento de Matemática, Carretera Camajuaní Km 5.5, CP - 54830 Santa Clara, Villa Clara, Cuba; E-Mail: gonzalo@udv.etecsa.cu

1 Introducción

Actualmente está bien determinada la importancia de la optimización como uno de los principios básicos del análisis de los problemas complejos de decisión, que implican la selección de valores para cierto número de variables interrelacionadas, centrando la atención en un objetivo diseñado para cuantificar el rendimiento y medir la calidad de la decisión. Este objetivo se maximiza, o minimiza teniendo en cuenta las restricciones que pueden limitar la selección de los valores de las variables de decisión.

Una primera etapa es la elaboración del modelo matemático, donde se realiza el planteamiento del problema de optimización correspondiente. La segunda etapa se refiere a la selección y aplicación de los métodos, algoritmos y herramientas de cálculo necesarios para la búsqueda de la solución del problema matemático y en la tercera etapa se realiza la interpretación y validación práctica de los resultados obtenidos. La solución de los problemas, en la segunda etapa, es el contenido fundamental de la Programación Matemática. Generalmente, en los softwares de optimización no lineal, para lograr efectividad en clases más amplias de funciones se combinan varios algoritmos.

OPTIMIZA es un programa escrito en Borland Delphi v. 3.0 para resolver problemas de optimización del tipo

$$f(x) \to \min \ (\max) \tag{1}$$

sujeto a:

$$\begin{aligned} h_i(x) &= 0, \quad i = 1, 2, \ldots, m \\ g_j(x) &\leq 0, \quad j = 1, 2, \ldots, p \end{aligned}$$

donde $x \in S \subset \mathbb{R}^n; f : \mathbb{R}^n \to \mathbb{R}; h : \mathbb{R}^n \to \mathbb{R}^m; g : \mathbb{R}^n \to \mathbb{R}^p$, basado en los métodos de Lagrangiano Aumentado [1] para transformar el problema original en otro sin restricciones aplicando entonces a éste Métodos quasi-Newton [1], [2] para obtener una solución. Para obtener una aproximación a la inversa del Hessiano de la función, se utilizan algoritmos del tipo $L - BFGS$ [3].

En la siguiente sección, se muestran los métodos utilizados en OPTIMIZA. En la sección 3 se presenta la estructura general del software. En la sección 4 se describe como utilizarlo y en la sección 5 se relacionan algunos de los experimentos numéricos realizados y se comparan los resultados obtenidos con la solución de los problemas prueba del Generador Schittkowski [4].

2 El método

Los problemas generales de la programación no lineal (1) poseen una alta complejidad para solucionarlos. Es por ello que se busca transformarlos en problemas similares que tengan la misma solución, pero de complejidad menor.

2.1 Conversión del problema a uno sin restricciones

En la bibliografía el tipo de conversión que más se trata son los métodos de Penalización, donde se propone que se transforme el problema original (1) por otro sin restricciones,

donde a la función objetivo se le adiciona un término que prescribe un alto costo por la violación de las restricciones, es decir:

Dado el problema (1), obtener

$$f(x) + cP(x) \to \min$$

donde $c > 0$ y P es una función en $\mathbb{R}^n$ que satisface:

1. P es continua
2. $\forall x \in \mathbb{R}^n$, $P(x) \geq 0$
3. $P(x) = 0$ si, y solo si, $x \in S$

Por ejemplo, si S está definido por cierto número de restricciones de desigualdad

$$S = \{x : g_j(x) \leq 0 \ j = 1, 2, ..., p\}$$

una función de penalización muy útil es la definida por

$$P(x) = \frac{1}{2} \sum_{j=1}^{p} (\max[0, g_j(x)]^2$$

con lo cual se garantizan las condiciones exigidas para p y además su continuidad y diferenciabilidad si las funciones $g_j, j = 1, 2, ..., p$ lo son.

Por otra parte, una de las clases generales de métodos de programación no lineal más efectivos es la de los métodos de Lagrangiano Aumentado [1], también llamados Métodos de los Mutiplicadores. Estos métodos se pueden considerar como métodos combinados de funciones de penalización y de dualidad local.

En el método de Lagrangiano Aumentado para resolver problemas del tipo

$$f(x) \to \min \tag{2}$$

$$\text{sujeto a } h(x) = 0$$

se toma la función $f(x) + \lambda^t h(x) + \frac{1}{2} c \left|h(x)\right|^2$, donde $x \in \mathbb{R}^n, c \in \mathbb{R}^+, \lambda \in \mathbb{R}^m$.

Desde el punto de vista de una función de penalización, el Lagrangiano Aumentado, para un valor fijo del vector λ, es simplemente la función de penalización cuadrática estándar para el problema

$$f(x) + \lambda^t h(x) \to \min \tag{3}$$

$$\text{sujeto a } h(x) = 0.$$

Se observa que el problema (3) es equivalente al problema (2), pues las combinaciones de las restricciones asignadas a $f(x)$ no alteran el punto mínimo ni el valor mínimo.

El Lagrangiano Aumentado se considera como una función de penalización exacta cuando se utiliza el valor adecuado de λ^* y se demuestra en [1] que una buena aproximación a la incógnita λ^* es $\lambda_{k+1} = \lambda_k + ch(x_k)$.

Desde el punto de la teoría de la dualidad, el Lagrangiano Aumentado es simplemente el Lagrangiano estándar para el problema

$$f(x) + \frac{1}{2}c\,|h(x)|^2 \to \min \tag{4}$$

$$\text{sujeto a } h(x) = 0.$$

Este problema equivale al problema original (2), pues la suma del término $\frac{1}{2}c\,|h(x)|^2$ al objetivo no cambia el valor optimal, el punto solución óptimo, ni el multiplicador de Lagrange. Sin embargo, mientras que el Lagrangiano original puede no ser convexo cerca de la solución, el término $\frac{1}{2}c\,|h(x)|^2$ tiende a hacer convexo el Lagrangiano. De hecho, para c suficientemente grande el Lagrangiano será localmente convexo.

Como en el problema (2) se tratan solo ecuaciones de igualdad habrá que realizar una pequeña transformación a las restricciones de desigualdad del problema original (1). Sea el problema

$$f(x) \to \min$$

$$\text{sujeto a } g(x) \leq 0,$$

donde $f : \mathbb{R}^n \to \mathbb{R}, g : \mathbb{R}^n \to \mathbb{R}^p, x \in \mathbb{R}^n$. Se supone que este problema tiene una solución bien definida x^*, que es un punto regular de las restricciones que satisface las condiciones de suficiencia de segundo orden para un mínimo local. Este problema se puede formular como un problema equivalente con restricciones de igualdad:

$$f(x) \to \min \tag{5}$$

$$\text{sujeto a } g_j(x) + z_j^2 = 0, \quad j = 1, 2, ..., p.$$

Haciendo un ánalisis de como determinar z, en [1] se llega a la siguiente expresión:

$$z_j^2 = \begin{cases} -g_j(x) - \frac{\mu_j}{c} & \text{si } -g_j(x) - \frac{\mu_j}{c} > 0 \\ 0 & \text{en el resto de los casos.} \end{cases}$$

O, de otra forma, $z_j^2 = \max\left[0, -g_j(x) - \frac{\mu_j}{c}\right]$ quedando el Lagrangiano para el caso de restricciones de desigualdad en la forma

$$L_c = f(x) + \sum_{j=1}^{p} \frac{1}{2}c\left\{\left[\max\left(0, \mu_j + cg_j(x)\right)\right]^2 - \mu_j^2\right\}.$$

Entonces para el problema (5) el multiplicador de Lagrange μ se puede ajustar igual que en el caso de igualdad.

2.2 Solución del problema sin restricciones

Para resolver el problema sin restricciones, en [2] se hace referencia a diversos algoritmos de optimización no lineal sin restriciones que han sido los más utilizados hasta ese momento. En particular, los métodos quasi-Newton comienzan tomando un punto inicial x_0

y van generando una sucesión de puntos que converjan al mínimo o máximo buscado. La búsqueda se hace de la siguiente manera:

$$x_{k+1} = x_k + \alpha_k d_k, d_k = -H_k q_k, q_k = \nabla L_c(x_k)$$

donde: H_k es una aproximación a la inversa del Hessiano de la función, y α_k es el resultado de la búsqueda lineal para encontrar el mínimo de la función en la dirección de descenso.

Son varios los métodos para calcular la aproximación a la inversa del Hessiano pero, como se aprecia en [2], donde se hace una comparación entre varios de estos métodos, se tiene a los algoritmos de la familia Broyden, y en especial a los $BFGS$, como los candidatos más favorecidos.

Lo que el algoritmo propone realizar es una actualización de la forma:

$$H_{k+1}^{BFGS} = V_k^t H_k V_k + \rho_k s_k s_k^t,$$

donde $s_k = x_{k+1} - x_k$, $y_k = g_{k+1} - g_k$, $\rho_k = \frac{1}{y_k^t s_k}$, y $V_k = I - \rho_k y_k s_k^t$.

Los métodos quasi-Newton se suelen ejecutar de manera continua, comenzando con una aproximación inicial y mejorándola sucesivamente mediante el proceso iterativo. Bajo ciertas condiciones algo estrictas se puede demostrar que este procedimiento es globalmente convergente [1]. Por otro lado, si los métodos quasi-Newton se inician de nuevo cada n ó $n+1$ pasos volviendo a hacer la aproximación a la inversa del Hessiano igual a su valor inicial, la convergencia global se garantiza mediante la presencia del primer paso descendente de cada ciclo (que actúa como un paso espaciador).

El algoritmo para solucionar el problema sin restricciones nos quedaría de la forma siguiente:

Paso 0. Dado x_0 y H_0 matriz definida positiva; $k = 0$

Paso 1. Calcular

$$\begin{aligned} d_k &= -H_k q_k \\ x_{k+1} &= x_k + \alpha_k d_k \end{aligned}$$

donde α satisface las condiciones de Wolfe:

$$\begin{aligned} f(x_k + \alpha_k d_k) &\leq f(x_k) + \beta' \alpha_k q_k^t d_k \\ q(x_k + \alpha_k d_k) &\geq \beta q_k^t d_k \end{aligned} \tag{6}$$

y se propone comenzar con $\alpha = 1; 0 < \beta' < \beta < 1$.

Paso 2. Calcular $q_{k+1} = q(x_{k+1})$

Paso 3. Verificar, si $|q_{k+1}| < \varepsilon$ entonces fin,
si no calcular H_{k+1} e ir al Paso 1.

Este método necesita $\mathcal{O}(n^2)$ localizaciones de memoria, mientras que, como se muestra en [3], una variante con las mismas propiedades de convergencia que el anterior, denominado $L-BFGS$, propone tomar la matriz de actualización como:

$$\begin{aligned} H_{k+1}^{L-BFGS} &= \left(V_k^t ... V_{k-\widehat{m}}^t\right) H_0 \left(V_{k-\widehat{m}} \cdots V_k\right) + \\ &+\rho_{k-\widehat{m}} \left(V_k^t \cdots V_{k-\widehat{m}+1}^t\right) s_{k-\widehat{m}} s_{k-\widehat{m}}^t (V_{k-\widehat{m}+1} \cdots V_k) + \\ &+\rho_{k-\widehat{m}+1} \left(V_k^t \cdots V_{k-\widehat{m}+2}^t\right) s_{k-\widehat{m}+1} s_{k-\widehat{m}+1}^t (V_{k-\widehat{m}+2}^t \cdots V_k) + \cdots + \rho_k s_k s_k^t. \end{aligned} \tag{7}$$

Hay que notar que la matriz H_k no se forma explícitamente, sino que los $\widehat{m}+1$ valores de y_j y s_j son almacenados separadamente y con ellos se realiza la actualización. Lo que se propone es actualizar m pares de datos y hacer la actualización de la matriz H_{k+1} con solo esta información, pero se obtiene una matriz semejante en cuanto a resultados numéricos a la del $BFGS$.

Para seleccionar el valor de m, es decir, la cantidad de datos que se almacenarán, se demuestra en [3] que con base en las pruebas numéricas realizadas una buena selección es tomar $m = 5$, y que tomar valores mayores no incrementa realmente el rendimiento del algoritmo. El Paso 3 nos quedaría entonces de la forma:

Paso 3. Calcular $\widehat{m} = \min\{k, m-1\}$.
Actualizar H_0 en $\widehat{m}+1$ veces usando los pares $\{y_j, s_j\}_j^k = k - \widehat{m}$
H_{k+1}^{L-BFGS} se actualiza de acuerdo con (7)

Hay que notar que esta actualización es muy fácil de implementar y es similar en complejidad al $BFGS$, con la diferencia en el cálculo de la matriz.

2.3 Búsqueda Lineal

Ya encontrada la dirección de descenso en la cual la función objetivo decrece, se debe buscar el valor por el cual se debe multiplicar el vector de dirección d para lograr un decrecimiento efectivo. Esta α se pudiera calcular prácticamente exacta, pero habría que pagar un alto costo computacional y la correspondiente pérdida de tiempo, es por esto que se ha desarrollado toda una teoría de como realizar la búsqueda lineal imprecisa, pero que garantice que la función decrezca. Diferentes métodos relacionados con lo anterior se estudian en [1]-[3] y todos coinciden en buscar α que no sea demasiado grande o pequeña, o sea, que debe satisfacer las condiciones de Wolfe (6).

En [4] se propone que se realice esta búsqueda comenzando con $\alpha = 1$ y se verifican las desigualdades (6).

El algoritmo para determinar la amplitud del paso nos quedaría de la forma siguiente:

Paso 0. Tomar $\alpha = 1, \beta \in (0,1)$

Paso 1. Verificar las condiciones de Wolfe (6)

Paso 2. Si no se cumplen, tomar $\alpha = \frac{1}{2}\alpha$,
asignar $\alpha_k = \alpha$ e ir al Paso 1,
en caso contrario $x_{k+1} = x_k + \alpha_k d_k$

En el presente trabajo se utilizan los valores $\beta' = 10^{-4}$ y $\beta = 0.9$ los cuales son recomendados en [2], [5], [6].

2.4 Escalas para mejorar la convergencia del algoritmo

Es conocido que una simple variación de la escala de las variables puede provocar una mejora considerable en el comportamiento del algoritmo, por lo que pasaremos a realizar algunas observaciones en este sentido.

En [3] se plantean cuatro tipos de escalas para mejorar la matriz H_k y las propuestas son:

- Escala M1: $H_k^{(0)} = H_0$ (no escalada)
- Escala M2: $H_k^{(0)} = \gamma_0 H_0$ (solo un escalado inicial)
- Escala M3: $H_k^{(0)} = \gamma_k H_0$
- Escala M4: Igual a M3 durante las m primeras iteraciones.
  Para $k > m, H_k^{(0)} = D_k$, donde $\gamma_k = \frac{y_k^t s_k}{\|y_k\|^2}$ y $D_k = diag(d_k^i)$. En el trabajo de referencia se realizaron varios experimentos, obteniéndose los mejores resultados con la escala M3.

Algoritmo Final de Solución

Dados f, g, h

Paso 0. Inicializar $\lambda, \mu, c, H_0, x_0$

Paso 1. Calcular $L_c, q_k = \nabla L_c(x_k)$

Paso 2. Si $|q_k| < \varepsilon$, entonces

$$\begin{cases} \text{Si } |h(x)| < \varepsilon, \text{ entonces } \begin{cases} \text{si } g(x) \leq \varepsilon, \text{fin} \\ \text{si } g(x) > \varepsilon, \text{ actualizar } \mu \end{cases} \\ \text{en caso contrario, actualizar } \lambda \text{ y si } g(x) > \varepsilon, \text{actualizar } \mu \end{cases}$$

Actualizar L_c

Paso 3. Calcular un nuevo punto

$$d_k = -H_k q_k, \; x_{k+1} = x_k + \alpha_k d_k$$

Hallar q_{k+1} e ir al Paso 2.

3 Estructura de OPTIMIZA

3.1 El compilador

Unit Scanner

En esta `unit` se define la función `Scan`, a la cual se le pasa como parámetro una cadena de caracteres que es dividida en `tokens` de acuerdo con los símbolos divisores que se especifican en :

`Nestid:` símbolos que indican anidamiento de estructuras y

`Primary:` símbolos divisores primarios.

Los `tokens` son clasificados en varios tipos y devueltos en una lista por la función.

Unit Parser

Esta `unit` contiene la clase `TParser`, la cual exporta un conjunto de métodos que a partir de una lista de `tokens` verifican que estos se encuentren en un correcto orden sintáctico de acuerdo con las reglas gramaticales definidas y construyen una representación interna del objeto específico que se trate, devolviendo una referencia a dicho objeto. Los métodos son los que se presentan a continuación:

```
function GetRelation(aList: TList): TRelation;
function GetEquation(aList: TList): TEquation;
function GetExpression(aList: TList): TExpression;
function GetIdentifier(aList: TList): TIdentity;
function GetFunction(aList: TList): TUserFunction;
```

Unit Constants

Todas las constantes usadas en la interfaz con el usuario son definidas en esta `unit`. Se definen los identificadores de las funciones matemáticas y los signos de las relaciones algebraicas.

3.2 El núcleo matemático

Unit MathObject

El objeto matemático, `TMathObject`, es la clase base de todas las que conforman el núcleo matemático. A partir de esta clase se especializan las siguientes:

`TNumber`

`TExpression`

`TMatrix`

`TRelation`

La clase `TNumber` encapsula a un número real, a través de la propiedad `Value` se accede a su valor.

Unit Struct

Una de las principales ramas de nuestra jerarquía de clases se encuentra definida en esta `unit` a partir de `TExpression`, base de todo el andamiaje matemático de esta aplicación.

Una expresión matemática puede ser una constante `(TConstant)`, o una variable o constante literal `(TIdentity)` o una concatenación de cualesquiera de ellas enlazadas a través de operaciones matemáticas. Estas operaciones pueden ser binarias `(TBinaryOperation)` o unarias `(TUnaryOperation)` dependiendo de la cantidad de argumentos sobre los que actúan.

Toda expresión se puede evaluar a través de la propiedad `Value`, y mediante los métodos:

```
function Derivative(Respect: TIdentity): TExpression;
procedure Simplify;
```

se puede derivar con respecto a una variable y se puede simplificar respectivamente una expresión.

Las operaciones unarias tienen la propiedad `Child,` que es la expresión sobre la cual se realiza la operación; mientras que las binarias tienen las propiedades `LeftChild` y `RightChild`. Estas propiedades son renombradas en las clases descendentes de acuerdo a sus características.

De `TUnaryOperation` heredan las siguientes clases:

`TNeg(X);` es el negativo del argumento, $-$`X`

`TInv(X);` es el inverso del argumento, $\frac{1}{X}$

`TExp(X);` es el exponencial de `X`,e^x

`TNeperian(X);` es el logaritmo neperiano de `X`, `Ln X`

Además heredan también las funciones trigonométricas y sus inversas:

`TSin, Tcos, TTan, TCot, TArcSin, TArcCos, TArcTan;` y las funciones hiperbólicas y sus inversas: `TSinh, TCosh, TTanh, TCoth, TArcSinh, TArcCosh, TArcTanh.`

De `TBinaryOperation` heredan las siguientes clases:

`TSum (X,Y);` es la suma de `X+Y`

`TProd (X,Y);` es el producto de `X*Y`

`TPower (X,Y);` es `X` elevado a la potencia `Y`.

`TRoot (X,Y),` es la raíz `Y`-ésima de `X.`

Se define en esta `unit` además las clases de matrices `TPointMatrix` y `TPointVector,` especializadas para manipular variables `(TIdentity).`

Unit Matrix

Las propiedades y el comportamiento básico de las matrices son definidas en esta `unit` a través de la clase `TMatrix.`

Nuestras matrices pueden contener elementos de cualquier tipo de objeto matemático, pero todos los elementos que contenga una matriz tienen que ser de la misma clase y dicha clase se especificará en la propiedad `ElementType.` Para describir la cantidad de filas y columnas de una matriz se utlizan las propiedades `RowsCount` y `ColumsCount`

respectivamente y para acceder a un elemento específico se hace a través de la `property Elements[const row, col:integer]:TMathObject;`

También se define en esta `unit TVector,` clase que hereda la estructura de `TMatrix` y especializa su interfaz para hacerla más cercana a los términos matemáticos. Así incorpora una nueva propiedad, `VectorType,` para describir si se trata de un vector fila o columna.

Una matriz especializada en contener números es `TNumericMatrix,` que además de tener el mismo comportamiento de su ancestro `TMatrix` incorpora nuevos métodos para realizar operaciones entre matrices numéricas como la multiplicación, la adición y la multiplicación por un escalar. También se define `TNumericVector.`

Unit Relations

En esta `unit` se define la clase `TRelation`, que nos permite establecer relaciones de dos tipos. El primero son las funciones explícitas definidas por los usuarios que relacionan a un identificador con una expresión algebraica mediante la clase `TUser-Function.` El segundo es la relación que se puede establecer entre expresiones: mayor que, menor que, igualdad, etc, a través de la clase `TAlgebricRelation.`

En la interfaz de `TRelation` se encuentran las propiedades `LeftExpression` y `RightExpression` que son evidentemente las expresiones que se relacionan; el signo que establece la relación se escribe en `Sign`, y el método `Simplify` trata de hacer más simple las expresiones relacionadas.

Las principales propiedades de `TUserFunction` son:

`Identifier:` es el nombre que identifica a la función.

`Expression:` es el cuerpo mismo de la función.

`Variable[i]:` es la variable i-ésima de la función.

`VariablesCount:` es la cantidad de variables que están presentes en la expresión de la función.

El método

`function EvaluateAt(Point: TPointVector):TNumber;`

brinda un mecanismo para evaluar la función en un punto arbitrario `Point.`

La clase `TAlgebricRelation` se especializa en `TEquation` y `TInequation,` es decir, ecuaciones e inecuaciones.

En esta `unit` se define además `TFunctionalMatrix` y `TFunctionalVector` ambas descendientes de `TMatrix` y especializadas en contener funciones. Estas clases de matrices se pueden evaluar en un punto a través del método

`function EvaluateAt(Point:TPointVector):TNumericMatrix;`

devolviendo una matriz numérica.

3.3 El problema a resolver

Unit Problem

En esta `unit` se encuentra la clase `TProblem` la cual se encarga de encapsular el problema de optimización que se trata de resolver y que tiene la forma:

$$f(x) \rightarrow (\max, \min)$$

$$\text{sujeto a } h(x) = 0, g(x) < 0.$$

En su interfaz encontramos las siguientes propiedades:
`ObjetiveFn`: Es la función que será objeto de la optimización.
`SolveObjetive`: Indica si el problema se quiere maximizar o minimizar.
`Constraint[i]`: Es la restricción i-ésima.
`ConstraintsCount`: Indica la cantidad de restricciones que tiene el problema.
`Variable [i]`: Permite acceder a la variable i-ésima.
`VariablesCount`: Es la cantidad de variables que contiene el problema.
`Defined[i]`: Permite acceder al identificador de la expresión auxiliar i-ésima.
`Definition[i]`: Es la expresión auxiliar i-ésima
`DefinitionsCount`: Indica la cantidad de expresiones auxiliares que se han definido en el problema.
`Gradient:` Es el gradiente de la función objetivo.

Además de estas propiedades `TProblem` tiene los métodos:

```
procedure AddConstraint(aConstraint:  TAlgebricRelation);
procedure AddDefinition(aDefinition: TUserFunction);
```

que se utilizan para añadir restricciones y añadir expresiones auxiliares al problema respectivamente.

3.4 Los métodos de solución

Unit Methods

En esta `unit` se encuentran la clase básica `TSolveMethod` que encapsulará cualquier algoritmo para la solución del problema a optimizar, a su vez, la clase `TLinealSearch` encapsula los algoritmos de búsqueda lineal.

`TSolveMethod` exporta la función `Resolve` que como parámetro recibe el problema a optimizar y devuelve el punto donde dicho problema es óptimo.

La clase `TLBFGSMethod` es la implementación del algoritmo de optimización de modelos `BFGS` de memoria limitada.

4 El uso de OPTIMIZA

4.0.1 Uso de la interfaz gráfica

La interfaz de usuario que utiliza el paquete Optimiza es un esquema multipágina que es de gran comodidad ya que va guiando paso a paso al usuario en el proceso de entrar los datos de un problema y darle solución, especificando en cada momento la acción que debe realizarse y comprobando que está correcta antes de pasar al próximo paso.

En el borde inferior de la ventana principal se encuentra la barra de estado que brinda una breve orientación sobre la utilización de cada uno de los controles que aparecen en la aplicación. Para una explicación más detallada se puede solicitar el servicio de ayuda presionando la tecla F1 o buscar en el submenú Help o hacer click sobre los botones de ayuda.

Cuando se comienza la aplicación, la primera página que aparece activa lleva el título `General`. En esta página aparece un conjunto de atributos del problema que, opcionalmente, se puede ignorar:

- `Name`: Aquí escribirá el nombre del problema.
- `Classification`: Hace referencia a alguna clasificación del problema.
- `Source`: Es la fuente de origen del problema.
- `Comments`: Cualquier comentario que desee hacer sobre el problema.

Hay también en esta página dos incisos que sí es obligatorio especificar: `Variables` y `Constraints` que significan el número de variables y el número de restricciones que respectivamente contiene el problema.

Para editar cualquiera de estos incisos debe hacer click en el panel que aparece a la derecha de la etiqueta que lo identifica.

La próxima página titulada `Functions` permite la edición de la función que desee optimizar y de funciones auxiliares que podrá utilizar para hacer más cómoda y comprensible la edición de las expresiones que utilice.

Debajo del texto `Objetive function` aparece un botón que puede tener dos estados `min` minimizar y `max` maximizar la función objetivo, para conmutar el estado haga un click sobre él. A la derecha de este botón hay una caja para editar la función objetivo.

Bajo el texto `Definitions` se puede editar una lista de funciones auxiliares haciendo click sobre los paneles que aparecen al mover el puntero del mouse sobre esa región.

La página siguiente se denomina `Constraints` y es la que permite editar las restricciones del problema.

La siguiente página `Start` contiene una lista de las variables del problema, al lado de cada una de ellas se escribirá el valor inicial que tomarán en el momento de aplicar el algoritmo de solución del problema. El valor inicial implícito es cero.

La última página `Solution` es la que mostrará los resultados de aplicar el algoritmo de solución al problema planteado. Al seleccionar esta página se echa a andar el mecanismo de cálculo y luego se podrán observar los resultados del mismo.

OPTIMIZA brinda también la posibilidad de trabajar con archivos fuente. Es decir, usted puede escribir con ayuda de un editor de texto un archivo en formato texto MS-DOS que contenga las especificaciones necesarias para optimizar el modelo de un problema (vea en el punto 4.3 el formato de los ficheros fuente) y luego el paquete será capaz de leerlo y ejecutarlo.

4.0.2 Gramática de las expresiones matemáticas

Las expresiones matemáticas se definen como una constante numérica o una constante literal o una variable o una concatenación de cualesquiera de ellas enlazadas a través de operaciones matemáticas como suma, resta, multiplicación, división, funciones de potencia, funciones trigonométricas y otras.

La tabla siguiente ilustra la sintaxis en la escritura de cada uno de los elementos que constituyen las expresiones matemáticas.

Expresión	**Sintaxis**	**Ejemplos**
Constante numérica	⟨número⟩	34,5.78
Constante literal	⟨texto⟩	total,delta
Variable	X⟨número⟩	X1,X2
Suma	⟨expresión⟩+⟨expresión⟩	X3+27,X1+X2
Producto	⟨expresión⟩*⟨expresión⟩	2*X4,X2*X3
Resta	⟨expresión⟩-⟨expresión⟩	-X5,3*X1-X2
División	⟨expresión⟩/⟨expresión⟩	X1/4,(X2+X3)/X1
Potencia	Power(⟨expresión⟩,⟨expresión⟩)	Power(2*X1+X2,X0)
Exponencial	Exp (⟨expresión⟩)	Exp X1
Logaritmo neperiano	Ln(⟨expresión)	Ln X1
Seno	Sin (⟨expresión⟩)	Sin X1
Coseno	Cos (⟨expresión⟩)	Cos X1
Tangente	Tan (⟨expresión⟩)	Tan X1
ArcSeno	ArcSin (⟨expresión⟩)	ArcSin X1
ArcCoseno	ArcCos (⟨expresión⟩)	ArcCos X1
ArcTangente	ArcTan (⟨expresión⟩)	ArcTan X1
Seno hiperbólico	Sinh (⟨expresión⟩)	Sinh X1
Coseno hiperbólico	Cosh (⟨expresión⟩)	Cosh X1
Tangente hiperbólica	Tanh (⟨expresión⟩)	Tanh X1
ArcSeno hiperbólico	ArcSinh (⟨expresión⟩)	ArcSinh X1
ArcCoseno hiperbólico	Arcosh (⟨expresión⟩)	Arcosh X1
ArcTang. hiperbólica	ArcTanh (⟨expresión⟩)	ArcTanh X1

4.0.3 Formato de los ficheros fuente

Los ficheros fuente de OPTIMIZA deberán ser escritos siguiendo las siguientes reglas:

- El fichero debe ser un fichero texto del formato MS-DOS.
- No hay diferencias entre las letras mayúsculas y minúsculas.
- Debe comenzar con la palabra PROBLEMA seguida de un texto entre comillas que se tomará como el título.
- Cualquier texto precedido de "//" será ignorado.
- Cualquier texto precedido de "/$" será tomado como el comentario que se visualiza en la segunda pantalla.
- Antes de hacer referencia a una variable debe aparecer la claúsula VARIABLES seguida del número de variables que usará, no se permite hacer referencias a variables de índice mayor que el especificado.
- Si quiere especificar el significado de una variable lo puede hacer digitando el nombre de la variable seguida de su significado entre comillas.

- La función objetivo se define como:
  `Max =` expresión matemática ó
  `Min =` expresión matemática.

- La forma de representar las restricciones es:
  Expresión matemática = Expresión matemática
  Expresión matemática ⟨Expresión matemática
  Expresión matemática ⟩Expresión matemática `NoNeg`(variable inicial,variable final).

- Las especificaciones deben terminar con la claúsula FIN y cualquier texto que aparezca después será ignorado.

5 Experimentos numéricos

OPTIMIZA fue validado utilizando los ejemplos-prueba que se presentan en el generador Schittkowski [4]. Estos problemas fueron resueltos en una PC 586 a 220 MHz con 32 Mb de memoria RAM y se utilizó una precisión numérica de 10^{-14}, obteniéndose la solución correcta en 80 problemas. A continuación se muestran los resultados obtenidos durante la ejecución del programa de algunos problemas, comparando al mismo tiempo con las soluciones que brinda el generador de problemas.

Problema 1

```
Variables 2
   //Función Objetivo
   MIN = 100*(X2-X1^2)^2+(1-x1)^2
   //Sujeto a:
   X2+1.5> 0
   X1, -2
   X2, 1
   Fin
   //Resultado...
   FuncObj= 7.48653177700473E-9
   X1, 0.9999381
   X2, 0.9998742
   //Parámetros...
   Tiempo de ejecución= 0:00:01
   Cantidad iteraciones= 98
   Evalaciones func.  Obj.= 397
   Evalac.  Der.  Func.  Obj.= 200
   Eval.  Prom.  Rest.= 398
   Eval.  Prom.  Der.  Rest.= 200
```

Respuesta del Generador: `(X1, X2)=(1,1)`

Problema 3

```
Variables 2
// Función Objetivo
MIN= X2+(10E-5)*((X2-X1)^2)
//Sujeto a:
X2> 0
X1, 10
X2, 1
Fin
//Resultado...
FuncObj= -3.18932098762654E-5
X1, 0.0008589
X2, -0.0000026
// Parámetros...
Tiempo de ejecución= 0:00:00
Cantidad de iteraciones= 38
Evalaciones func.Obj.= 141
Evalac. Der. Func. Obj.= 82
Eval. Prom. Rest.= 143
Eval. Prom. Der. Rest.= 82
```

Respuesta del Generador: `(X1, X2)=(0,0)`

Problema 4

```
Variables 2
   //Función Objetivo
   MIN= 1/3*(X1+1)^3+X2
   //Sujeto a:
   X1-1> 0
   X2> 0
   X1, 1.125
   X2, 0.125
   FIN
   //Resultado...
   FuncObj=2.6666250697976
   X1, 1.0000050
   X2, 0.0000039
   //Parámetros...
   Tiempo de ejecución= 0:00:01
   Cantidad de iteraciones= 139
   Evalaciones func.Obj.= 616
   Evalac.Der.Func.Obj.= 300
   Eval.Prom.Rest.= 313
```

```
   Eval.Prom.Der.Rest.= 150
```
Respuesta del Generador: `(X1, X2)=(1,0)`

Problema 5

```
Variables 2
   //Función Objetivo
   MIN=SEN(X1+X2)+(X1-X2)^2-1.5*X1+2.5*X2+1
   //Sujeto a:
   X1-4 < 0
   X1+1.5 > 0
   X2-3 < 0
   X2+3 > 0
   Fin
   //Resultado...
   FuncObj= -1.91322258616816
   X1, -0,5470162
   X2, -1.5467840
   //Parámetros...
   Tiempo de ejecución= 0:00:00
   Cantidad de iteraciones= 28
   Evalaciones func.Obj.= 114
   Evalac.  Der.  Func.Obj.= 60
   Eval.Prom.  Rest.= 28
   Eval.  Prom.  Der.  Rest.= 15
```
Respuesta del generador: $(x1, x2) = \left(-\frac{\pi}{3} + \frac{1}{2}, -\frac{\pi}{3} - \frac{1}{2}\right)$

Problema 6

```
Variables 2
   //Función Objetivo
   MIN=(1-X1)^2
   //Sujeto a:
   10*(X2-X1^2)=0
   X1, -1.2
   X2,1
   Fin
   //Resultado...
   FuncObj= 1.82602346861371E-12
   X1,0.9999990
   X2,0.9999979
   //Prámetros...
   Tiempo de ejecución= 0:00:01
   Cantidad de iteraciones= 141
```

```
   Evalaciones func.  Obj.= 607
   Evalac.  Der.Func.Obj.= 288
   Eval.Prom.Rest.= 609
   Eval.Prom.Der.Rest.= 288
```
Respuesta del Generador: `(X1,X2)=(1,1)`

Problema 31

```
Variables 3
   //Función Objetivo
   MIN=9*X1^2+X2^2+9*X3^2
   //Sujeto a:
   X1*X2-1 > 0
   X1 > -10
   X1 < 10
   X2 > 1
   X2 < 10
   X3 > -10
   X3 < 1
   X1,1
   X2,1
   X3,1
   Fin
   //Resultado...
   FuncObj= 5.99990285656424
   X1,0.5773535
   X2,1.7320450
   X3,-0.0000043
   //Parámetros...
   Tiempo de ejecución= 0:00:01
   Cantidad de iteraciones= 70
   Evalaciones func.Obj.= 309
   Evalac.Der.Func.Obj.= 222
   Eval.Prom.Rest.= 44
   Eval.Prom.Der.Rest.= 31
```
Respuesta del Generador: $(x1, x2, x3) = \left(\frac{1}{\sqrt{3}}, \sqrt{3}, 0\right)$

Problema 39

```
Variables 4
   //Función Objetivo
   MIN=-X1
   //Sujeto a:
   X2-X1^3-X3^2=0
```

```
   X1^2-X2-X4^2=0
   X1,2
   X2,2
   X3,2
   X4,2
   Fin
   //Resultado...
   FuncObj= -1.00007166675227
   X1,1.0000700
   X2,1.0001800
   X3,0.0000000
   X4,0.0000000
   //Parámetros...
   Tiempo de ejecución= 0:00:04
   Cantidad de iteraciones= 203
   Evalaciones func.Obj.= 897
   Evalac.Der.Func.Obj.= 860
   Eval.Prom.Rest.= 454
   Eval.Prom.Der.Rest.= 430
```

Respuesta del Generador: `(X1,X2,X3,X4)=(1,1,0,0)`

Problema 44

```
Variables 4
   //Función Objetivo
   MIN= X1-X2-X3-X1*X3+X1*X4+X2*X3-X2*X4
   //Sujeto a:
   8-X1-2*X2 > 0
   12-4*X1-X2 > 0
   12-3*X1-4*X2 > 0
   8-2*X3-X4 > 0
   8-X3-2*X4 > 0
   5-X3-X4 > 0
   NO NEG (1,4)
   Fin
   //Resultado...
   FuncObj= -15.00071153231709
   X1, -0.0000876
   X2, 3.0000680
   X3, -0.0000728
   X4, 4.0000490
   //Parámetros...
   Tiempo de ejecución= 0:00:05
   Cantidad de iteraciones= 250
```

```
   Evalaciones func.Obj.= 1340
   Evalac.  Der.  Func.  Obj.= 1040
   Eval.  Prom.  Rest.= 134
   Eval.  Prom.  Der.  Rest.= 104
```
Respuesta del Generador: (X1,X2,X3,X4)=(0,3,0,4)

Problema 46

```
Variables 5
   //Función Objetivo
   MIN= (X1-X2)^2+(X3-1)^2+(X4-1)^2+(X5-1)^6
   //Sujeto a:
   X1^2*X4+SEN(X4-X5)-1=0
   X2+X3^4*X4^2-2=0
   X1, 7.1
   X2, 1.75
   X3, 0.5
   X4, 2
   X5, 2
   Fin
   //Resultado...
   FuncObj= 3.08589497542169E-9
   X1, 0.9998090
   X2, 0.9998255
   X3, 1.0000380
   X4, 1.0000070
   X5, 0.9996321
   //Parámetros...
   Tiempo de ejecución= 0:00:04
   Cantidad de iteraciones= 199
   Evalaciones func.Obj.= 829
   Evalac.  Der.  Func.  Obj.= 1010
   Eval.  Prom.  Rest.= 415
   Eval.  Prom.  Der.  Rest.= 505
```
Respuesta del Generador: (X1,X2,X3,X4,X5)=(1,1,1,1,1)

Problema 60

```
Variables 3
   //Función Objetivo
   MIN= (X1-1)^2+(X1-X2)^2+(X2-X3)^4
   //Sujeto a:
   X1*(1+X2^2)+X3^4-4-3*SQRT(2)=0
   X1 < 10
```

```
X2 < 10
X3 < 10
X1 > -10
X2 > -10
X3 > -10
X1, 2
X2, 2
X3,2
Fin
//Resultado
FuncObj= 0.0325681895448708
X1, 1.1048590
X2, 1.1966740
X3, 1.5352620
//Parámetros...
Tiempo de ejecución= 0:00:05
Cantidad de iteraciones= 291
Evalaciones func.  Obj.= 2717
Evalac.  Der.  Func.  Obj.= 885
Eval.Prom.  Rest.= 388
Eval.  Prom.  Der.  Rest.=126
```

Respuesta del Generador:

```
(X1,X2,X3)=(1.104859024, 1.196674194, 1.5352262257)
```

6 Conclusiones

OPTIMIZA ha sido validado con una clase amplia de funciones obteniéndose buenos resultados, superiores incluso a los de otros paquetes semejantes, con un 92 % de efectividad.

En las estructuras de datos se utilizó un diseño Orientado a Objetos y la implementación de las clases se ajusta correctamente al problema planteado de forma que la aplicación es robusta y flexible.

La forma de intercambiar información entre el usuario y la máquina permite editar ficheros de forma natural; si se desea editarlo en el ambiente, este reduce la entrada de errores y le orienta al usuario cómo desarrollar su tarea.

Debido a la modularidad del diseño del sistema, la extensión de su programación y adición de nuevos algoritmos se hace sencilla. En particular, se trabaja en extender el alcance del sistema para resolver problemas no lineales en enteros.

References

[1] Luenberger, G. (1984) *Programación Lineal y No Lineal*. Addison-Wesley Iberoamericana, México.

[2] Nocedal, J. (1992) "Theory of algorithms for unconstrained optimization", *Acta Numerica* **1**: 199–222.

[3] Nocedal, J.; Liu Dong, C. (1989) "On the limited memory BFGS method for large-scale optimization", *Mathematical Programming* **45**: 503–528.

[4] Hock, W.; Schittkokski, K. (1981) *Test Examples for Nonlinear Programming Codes.* Lecture Notes in Economics and Mathematical Systems **187**, Springer, Berlin.

[5] Buckley; Lenir, A. (1983) "QN-like variable storage conjugate gradients", *Mathematical Programming* **27**: 103–119.

[6] Nocedal, J. (1980) "Updating quasi-Newton matrices with limited storage", *Mathematical Comput.* **35**: 773–782.