MANUAL DE USUARIO

OPTIMIZATION LANGUAGE

Por Alicia González y Ernesto García

Léxico y Sintaxis

PYNGO es un lenguaje de optimización que planea facilitar para el usuario la resolución de modelos simples y las operaciones de matrices.

La sintaxis de PYNGO se basa en:

- Funciones, en caso de tenerlas.
  El usuario puede crear funciones para realizar operaciones especiales para el modelo.
- Modelo
  En la parte del modelo, es donde se define el modelo a optimizar.
  Ya sea para maximizar ventas o minimizar costos.

Cada función, tanto las creadas por el usuario como la función principal, model, cuentan con tres bloques importantes para su desarrollo:

- VARS
  En esta parte se declaran las variables que se van a utilizar durante la función.
  Es importante mencionar que el lenguaje no permite realizar operaciones con variables no previamente declaradas en este bloque, así como que la declaración de variables no se puede realizar en ningún otro lugar del programa.
  La declaración de variables se realiza de la siguiente manera:

  ```
  TYPE : ID ;
  ```

  Ó en caso de ser variables dimensionadas :

  ```
  TYPE [DIMENSION][DIMENSION] : ID ;
  ```

- DATA
  En esta parte se les asignan valores a las variables previamente declaradas en vars.
  La asignación de variables se realiza de la siguiente manera:

  ```
  ID = VALOR ;
  ```

  Ó en caso de ser variables dimensionadas :

  ```
  ID = VALOR EN SUBINDICE 1, VALOR EN SUBINDICE 2…,
  VALOR EN SUBINDICE N ;
  ```

Después de estos dos bloques, las funciones pueden contener cualquier tipo de estatuto incluyendo:

- Escritura
- Ciclos
- Condiciones
- Expresiones

A continuación se muestran ejemplos de la sintaxis de los estatutos anteriormente mencionados:

- Escritura
  ```
  print 5 + 10 ;
  ```
- Ciclos
  ```
  For i in .idarreglo {
    Print i;
  }
  ```
  NOTA: El idarreglo, debe ser el identificador de un arreglo previamente declarado.
- Condiciones
  ```
  If (4 > a) {
    Print i+1;
  }
  ```
- Expresiones
  ```
  Print 5 * 2 * 2 + 4 ;
  ```

Optimización

Y el modelo debe contener su función de optimización, y cualquier estatuto de los previamente mencionados.
El formato para la función de optimización es el siguiente :

```
MIN ó MAX {
    for i in .oferta_plantas {
        for j in .demanda_ciudades {
            build costos[i][j]*x[i][j];
        }
    }
   #Aquí se muestran las restricciones para la optimización
    where {
        for i in .oferta_plantas {
            for j in .demanda_ciudades {
                condition x[i][j] <= oferta_plantas[i];
            }
        }
        for j in .demanda_ciudades {
            for i in .oferta_plantas {
                condition x[i][j] >= demanda_ciudades[j];
            }
        }

    }
}
```

Operaciones con Matrices

PYNGO trata de enfocarse en las operaciones con matrices.
Las operaciones disponibles para realizar entre matrices son:
En todas las operaciones se supone:

```
model {
            vars {
            int[2][2]: a, b;
            }

            data {
            a = 4, 1, 6, 10;
            b = 2, 3, 4, 5;
            }
```

…

- Suma de matriz con matriz
  `print a + b;`
- Suma de matriz con escalar
  `print a + 4;`
- Multiplicación de matriz por matriz
  `print a * b;`
- Multiplicación de matriz por escalar
  `print a * 4;`
- Resta de matriz con matriz
  `print a - b;`
- Resta de matriz con escalar
  `print a - 4;`
- Columna de una matriz
  `print col a, 1;`
- Columnas de una matriz
  `print cols a[1, 2];`
- Inversa de una matriz
  `print inverse a;`

En caso de tener más dudas, o preguntas, favor de contactar a Alicia González : Alicia.gonzalez.90@gmail.com

Se anexan ejemplos de programas:

Fibonacci

```
func int fib(int n) {
      vars {
      }
      data {
      }
      if n == 1 {
      return 1;
      }
      if n == 2 {
      return 1;
      }
      else {
      return fib(n-1) + fib(n-2);
```

```
	}
}
model {
	vars {
	 int: a;
	}
	data {
		a = 8;
	}
	print fib(a);
}
```

Factorial

Programa desarrollado por usuario

```
func int factorial(int n) {
	data {
	}
	if n == 0 {
	return 1;
	}
	else {
	return n * factorial(n-1);
	}
}

model {
	vars {
	 int: a;
	}
	data {
		a = 5;
	}
	print factorial(a);
}
```

Multiplicación de Matrices

Programa desarrollado por usuario

```
model {
	vars {
		int[2][2]: a, b;
	}

	data {
		a = 4, 1, 6, 10;
		b = 2, 3, 4, 5;
	}

	print a * b;
	print a + b;
	print a - b;
}
```