Métodos Numéricos

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico 2

Grupo 14

Integrante	LU	Correo electrónico
Dabbah, Julián	015/09	djulius@gmail.com
Dahlquist, Ariel	383/10	ariel.dahlquist@gmail.com
Garrone, Javier	151/10	javier3653@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Introducción

En este T.P. deberemos decidir si la estructura de un puente, con ciertas cargas dadas, es estable. En este caso, analizaremos la estructura de puentes *Prat Truss*, escribiendo las ecuaciones de fuerzas que se aplican sobre ciertas partes de la estructura. Luego resolveremos este sistema lineal y verificaremos que ninguna de los valores de fuerza obtenidos superen un límite determinado por los materiales de construcción. Para resolver el sistema utilizaremos el método de Eliminación de Gauss, al que aplicaremos algunas optimizaciones que permite este problema en particular, al igual que para la representación de la matriz en memoria.

Una vez que contemos con una manera eficiente de determinar las fuerzas que actúan sobre el puente dada una cierta configuración de cargas, consideraremos colocar pilares intermedios de concreto si alguna fuerza pusiera en peligro la seguridad del puente. Para esto, desarrollaremos una heurística que decida cuántos y dónde ubicarlos para minimizar el costo de construcción, dado que la inserción de estos pilares es altamente costosa.

1.1. Puentes *Pratt Truss*

Un esquema de puente *Pratt Truss* puede verse en la figura 1.

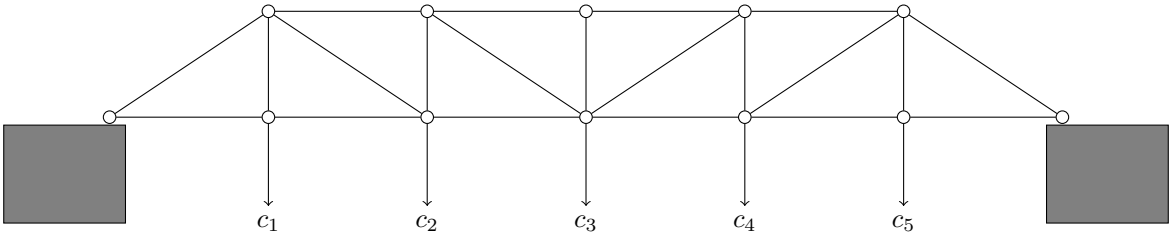


Figura 1: Esquema de la estructura de un puente *Pratt Truss*. (Cortesía Enunciado TP2)

Dado que realizaremos el análisis únicamente en dos dimensiones (supondremos que las cargas se distribuyen uniformemente en la profundidad), pensaremos de ahora en más en esquemas como éste. Llamaremos *link* a cada uno de los miembros de los que se compone la estructura, *junta* a los puntos donde éstos se unen y *sección* a la porción contenida entre dos links verticales sucesivos. Para el análisis en el T.P. trabajaremos con una cantidad par de secciones y supondremos que las cargas (y las fuerzas en general) se aplican únicamente sobre las juntas. Consideraremos como variables de la estructura la cantidad de secciones (n), la altura del puente (h), y la luz que debe cubrir (*span*). Podemos ver (como se muestraen la figura 2) que una estructura de n secciones cuenta con $2n$ juntas y $4n - 3$ links.

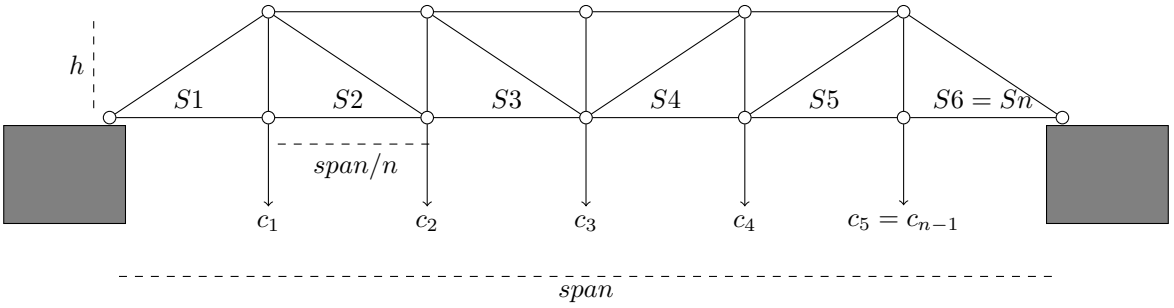


Figura 2: Variables a considerar de la estructura, $n = 6$. (Cortesía Enunciado TP2)

Para asegurar la estabilidad, requeriremos que la fuerza resultante sobre cada junta sea nula. Es decir, que las fuerzas horizontales y verticales (y la descomposición en estas direcciones de las fuerzas oblicuas) se anulen en cada junta. Para satisfacer esto, consideraremos, por cada junta, una ecuación en dirección horizontal y otra en vertical, cuyas incógnitas serán las fuerzas ejercidads por los links. Construyendo esto para todas las juntas, tendremos un sistema de $4n$ ecuaciones. Si consideramos una variable por la fuerza ejercida por cada link tendremos $4n-3$ variables, a las que agregaremos fuerzas verticales sobre las juntas de los extremos y una fuerza horizontal sobre uno de éstos, para modelar la interacción del terreno, tendremos un sistema cuadrado de $4n$. Observemos que la cantidad de fuerzas que se ejercen sobre cada link es una cantidad constante (y pequeña) en relación al tamaño de la estructura, con lo cual, la mayoría de los coeficientes de las incógnitas serán nulos, generando una matriz muy esparsa.

1.2. Representación Matricial

Dado que las variables involucradas en cada ecuación son pocas, la matriz asociada al sistema resulta con una gran cantidad de coeficientes nulos, lo que nos motivó a utilizar una estructura de datos que aproveche este hecho, antes que la implementación trivial como arreglo bidimensional. Más aún, veremos que la matriz resultante tiene estructura de bandas, y que nuestro algoritmo de eliminación la conserva, con lo cual durante toda la vida de la matriz, sigue siendo considerablemente esparsa, con lo cual las ventajas siguen siendo válidas. En esta representación, decidimos almacenar las filas como listas enlazadas ordenadas por índice creciente, conteniendo únicamente elementos no nulos. La matriz, entonces, resulta en un vector de n filas, en la cual, si bien el acceso aleatorio al elemento i, j es más costoso que en el caso del arreglo bidimensional, las operaciones por filas tienen el mismo costo que en aquél y evitan considerar operaciones entre valores nulos.

1.2.1. Matriz Banda

Formalmente, decimos que una matriz $A \in \mathbb{R}^{n\times n}$ tiene estructura de bandas p, q si $a_{i,j} = 0$ cuando $j \leq i - p$ ó $j \geq i + q$, $1 \leq i, j \leq n$.

Lema 1.2.1. *Sea $A \in \mathbb{R}^{n\times n}$, con estructura de bandas p, q. Luego, en el peor de los casos, en todos los pasos del algoritmo de Eliminación Gaussiana con pivoteo parcial, la matriz resultante tiene estructura de bandas $p, q + p$*

Demostración. Demostraremos el lema por inducción en la cantidad de pasos completados del algoritmo de Eliminación Gaussiana.
Sea $k = 1$, el primer paso del algoritmo.

Sea $i_1 = \arg\max_{1\leq i\leq n} |a_{i1}|$. Como A es matriz banda p, q, si $j = 1$ tenemos $a_{i,j} = 0$ cuando $1 \leq i - p \Leftrightarrow p + 1 \leq i$. Luego, i_1 no puede ser uno de éstos, con lo cual $i_1 \leq p$. Además, este argumento permite afirmar que el algoritmo únicamente modificará la fila i si $i \leq p$.

Consideremos las filas que se modifican. A lo sumo el último elemento no nulo de la fila i es el $a_{i,i+q-1}$ (llamémoslo $a_{i,j}$). Entonces, $j \leq q + i - 1 \leq q + p - 1$ (pues $i \leq p$).

Luego, al permutar la fila $k = 1$ con la fila i, tenemos que el último elemento no nulo de la fila 1 (ya permutada) está en la columna $j \leq q+i-1$, y como $i \leq p$, tenemos que el último elemento no nulo está en la columna $j \leq q+p-1$, con lo cual esta fila respeta la banda $p, q + p$. La fila i luego de la permutación, tiene su último elemento no nulo a lo sumo en la posición $j \leq q - 1$, con lo cual, como $1 \leq i \leq p$, $j \leq q - 1 + p$, lo que implica que esta fila también respeta la banda derecha $q + p$. Para el resto de las filas: de la definición de estructura de bandas se puede ver fácilmente que si $a \geq b$ y A tiene banda b, entonces también tiene banda a, con lo cual, luego del intercambio, toda la matriz tiene estructura de bandas $p, q + p$.
Además, tras aplicar la eliminación de la columna sólo se modifican las primeras p filas, con lo cual la matriz resultante tiene banda inferior $\leq p$ (observemos que la restricción sobre los elementos nulos sobre p no se puede aplicar en este caso pues resultaría $j \leq 0$). Para la banda derecha, observemos que en la fila resultante pueden aparecer elementos no nulos únicamente donde alguna de las dos filas involucradas en la elminiación tenía elementos no nulos. Sabemos que si estamos operando sobre la fila i, ésta puede tener elementos no nulos hasta la columna $j = q + p$ (en el caso de considerar la fila permutada, $j = q$ si no), lo mismo que la fila 1. Entonces, después de realizar los pasos de eliminación sobre todas estas filas, todas tienen su último elemento no nulo a lo sumo en la columna $p + q$; este valor produce la banda más ancha en la primera fila, cuando representa $p + q$ posiciones hacia la derecha de la diagonal, con lo cual, la matriz resultante tiene estructura de bandas $p, q+p$ luego de la primera iteración del algoritmo.

Hipótesis inductiva: Luego del paso k-ésimo del algoritmo de eliminación gaussiana (con $k < n$), la matriz resultante posee una estructura de bandas p', q' ($p' \leq p, q' \leq q + p$).

Paso inductivo: Apliquemos el paso $k + 1$. Del paso anterior obtuvimos, en el peor caso, una matriz banda $p, q + p$ por hipótesis inductiva. Además, sabemos que ninguna de las filas $i \geq p + k$ fue modificada en el paso anterior, pues el elemento a_{ik} es nulo. Luego, si consideramos la submatriz inferior derecha de A: A' de dimensión $k + 1 \times k + 1$, tenemos que A' es una matriz banda p, q y entonces, aplicando el mismo razonamiento que para $k = 1$, al aplicar Gauss obtendremos en el peor caso una matriz con bandas $p, q + p$ luego de esta iteración. □

1.3. Algoritmo de Resolución

Como dijimos, utilizaremos el método de Eliminación de Gauss, al que le agregaremos pivoteo parcial ya que no podemos asegurar que éste no sea necesario en las condiciones de la matriz. Más allá de esto, no realizamos mayores modificaciones al algoritmo, únicamente aprovechar la existencia de la banda inferior para reducir el alcance de la búsqueda del pivote, ya que por debajo de ésta sólo encontraría ceros.

2. Desarrollo

2.1. Construcción del sistema lineal

Dijimos que, dado un puente de n secciones, consideraríamos un sistema de $4n$ ecuaciones y $4n$ variables, dónde (casi todas) éstas representan las fuerzas que ejercen los links sobre las juntas. Para esto, debimos numerar los links de alguna manera, para asociarlos con las variables del sistema. Veamos cómo construimos esto, y qué podemos deducir de ello en términos de la matriz.

Veamos el ejemplo de numeración, para el caso $n = 6$, de la figura 3:

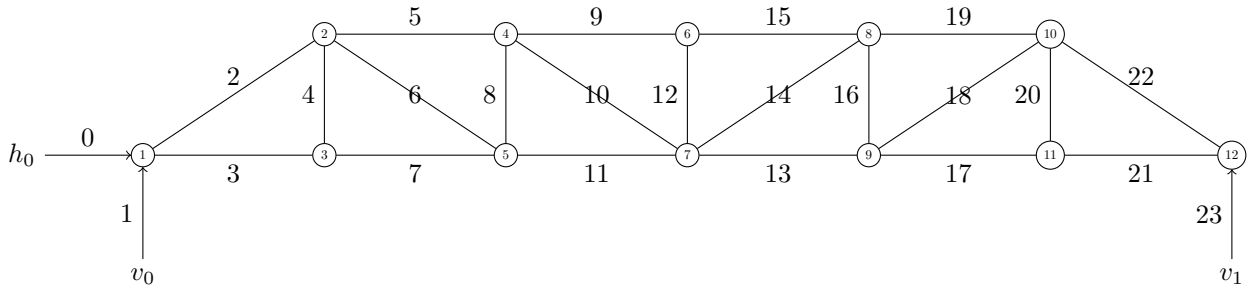


Figura 3: Numeración propuesta, en el caso $n = 6$.

Antes que nada, observemos que dado que la cantidad de secciones es par, la estructura del puente resulta simétrica, con lo cual podemos establecer las siguientes correspondencias entre las dos mitades:

- La fuerza i a la izquierda, se corresponde con la $4n - i$ a la derecha.
- La junta j a la izquierda arriba, se corresponde con la $2n - i$ a la derecha arriba.
- La junta j a la izquierda abajo, se corresponde con la $2(n + 1) - i$ a la derecha abajo.

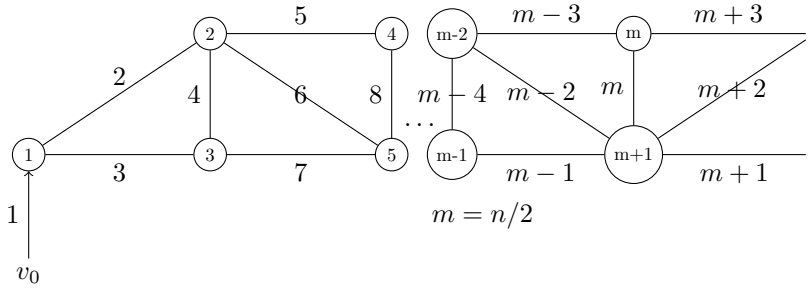
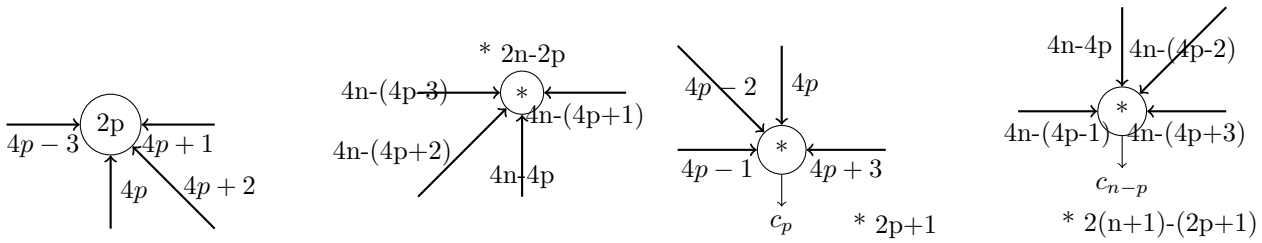


Figura 4: Numeración en el caso general, para la mitad izquierda.

Observemos que el caso complicado para la numeración de las fuerzas está dado por la cantidad variable (según n) de los links verticales interiores (es decir, aquellos que no son adyacentes ni a juntas consideradas extremos (1, 2 ó 3) ni a las juntas centrales. Las numeramos como $2p$ y $2p + 1$, con $2 \leq p < n/2$, lo que nos produjo los siguientes resultados, que exponemos en el caso general. Esto se puede verificar fácilmente si tomamos el primero de ellos para establecer la relación entre las fuerzas y las juntas, e incrementamos en 4 la numeración de cada fuerza tantas veces como links nos hayamos alejado del primero. Para la mitad derecha, resolvemos las relaciones de simetría explicadas anteriormente. Los cuatro casos posibles se muestran en la figura 5.

Veamos cómo se traduce lo anterior a las ecucaciones de un sistema en general. Para el desarrollo, supondremos que contamos con por lo menos cuatro secciones (el caso para $n = 2$ es un caso particular pequeño que no requiere tanta generalidad). Sabemos que necesitamos escribir dos ecuaciones por cada junta $1 \leq j \leq 2n$, y que cada una tiene su simétrica. Por lo tanto, proponemos utilizar las ecuaciones $2(j - 1)$, $2(j - 1) + 1$ si la junta está a la izquierda, y $4n-2j$, $4n-2j+1$ para su simétrica a derecha, siempre comenzando por la ecuación horizontal. Debimos descomponer la fuerza ejercida por los links oblicuos en las componentes verticales y horizontales, para lo que calculamos el seno y el coseno del ángulo con el que éstos inciden, simplemente como la proporción entre la longitud de la sección vertical (resp. horizontal) y la longitud del link oblicuo (lo que en las ecuaciones aparece como `s` y `c`, respectivamente). Además, supusimos que todos los links están en tensión (es decir, las fuerzas actúan siempre en el sentido hacia las juntas), con lo cual valores negativos indicarán que en realidad, en la estructura están en compresión; también decifinimos como sentido positivo hacia la derecha y hacia abajo. Luego, las figuras anteriores se traducen en:



(a) Junta superior izquierda (b) Junta superior derecha (c) Junta inferior izquierda (d) Junta inferior derecha

Figura 5: Juntas interiores

Para los extremos (juntas 1, 2 y 3 y sus simétricas):

$$
\begin{array}{lll}
0: & 0 = x_0 - \mathtt{c}x_2 - x_3 & 4n-2: \quad 0 = x_{4n-3} + \mathtt{c}x_{4n-2} \\
1: & 0 = -x_1 + \mathtt{s}x_2 & 4n-1: \quad 0 = \mathtt{s}x_{4n-2} + x_{4n-1} \\
 & & \\
2: & 0 = \mathtt{c}x_2 - x_4 - \mathtt{c}x_6 & 4n-4: \quad 0 = x_{4n-7} - x_{4n-3} \\
3: & 0 = -\mathtt{s}x_2 - x_4 - \mathtt{s}x_6 & 4n-3: \quad 0 = x_{4n-4} \\
 & & \\
4: & 0 = x_3 - x_7 & 4n-6: \quad 0 = \mathtt{c}x_{4n-6} + x_{4n-5} - \mathtt{c}x_{4n-2} \\
5: & -q_1 = x_4 & 4n-5: \quad -q_{n-1} = -\mathtt{s}x_{4n-6} - x_{4n-4} - \mathtt{s}x_{4n-2}
\end{array}
$$

Para el tramo del medio (juntas m y $m+1$):

$$
\begin{array}{rl}
2n-2: & 0 = x_{2n-3} - x_{2n+3} \\
2n-1: & 0 = -x_{2n} \\
 & \\
2n: & 0 = x_{2n-1} + \mathtt{c}x_{2n-2} - \mathtt{c}x_{2n+2} - x_{2n+1} \\
2n+1: & -q_{n/2-1} = x_{2n} + \mathtt{s}x_{2n-2} + \mathtt{s}x_{2n+2}
\end{array}
$$

Para las juntas interiores (juntas $2p$ y $2p+1$ con $2 \leq p < n/2$ y sus simétricas):

$$
\begin{array}{lll}
4p-2: & 0 = x_{4p-3} - x_{4p+1} - \mathtt{c}x_{4p+2} & 4(n-p): \quad 0 = \mathtt{c}x_{4(n-p)-2} + x_{4(n-p)-1} - x_{4(n-p)+3} \\
4p-1: & 0 = -x_{4p} - \mathtt{s}x_{4p+2} & 4(n-p)+1: \quad 0 = -\mathtt{s}x_{4(n-p)-2} - x_{4(n-p)} \\
 & & \\
4p: & 0 = \mathtt{c}x_{4p-2} + x_{4p-1} - x_{4p+3} & 4(n-p)-2: \quad 0 = x_{4(n-p)-3} + x_{4(n-p)+1} - \mathtt{c}x_{4(n-p)+2} \\
4p+1: & -q_p = \mathtt{s}x_{4p-2} + x_{4p} & 4(n-p)-1: \quad -q_{n-p} = x_{4(n-p)} + \mathtt{s}x_{4(n-p)+2}
\end{array}
$$

Ahora que tenemos las ecuaciones escritas en términos genéricos, miremos las bandas de la matriz. Podemos deducir los valores de las bandas de las cotas para las diferencias entre los índices de las variables y las ecuaciones (si no hubiera diferencia, la matriz sería puramente diagonal). La banda superior (o derecha) es 6 y se alcanza en la ecuación $2n-2$, mientras que la inferior (o izquierda) es 4 y se alcanza en las ecuaciones $4n-4$, $4p+1$, $4(n-p)+1$. [1]

2.2. Representación de la matriz asociada al sistema

Vimos, tal como lo habíamos adelantado en la introducción, que efectivamente la matriz asociada al sistema es una matriz con estructura de bandas, que el ancho de las bandas es constante con respecto al tamaño del sistema y que aún dentro de las bandas hay coeficientes nulos. En consecuencia, la matriz asociada al sistema es una matriz muy poco densa, con lo cual surge la idea de utilizar una representación que refleje esta propiedad para reducir el uso de memoria. La solución propuesta consiste en utilizar listas enlazadas para almacenar los elementos no nulos de cada fila. Dado que no almacenaremos los elementos nulos, cada elemento debe estar acompañado del índice que le corresponde dentro de la fila, y la lista ordenada por éstos de manera creciente, para que las operaciones entre filas se puedan realizar de manera eficiente. Las filas (listas) las almacenamos en un vector que se inicializa al crear la matriz, aún cuando éstas estén vacías.

[1] Observemos que si la diferencia entre el número de ecuación y el número de variable es p, la banda resultante es $p+1$, si pensamos que una matriz diagonal tiene bandas 1

2.2.1. Complejidad Espacial

Por lo que dijimos, es claro que si $nZ(A)$ es la cantidad de elementos no nulos de la matriz $A \in \mathbb{R}^{n \times n}$, en memoria tenemos reservadas $\mathcal{O}(zN(A)+n)$ posiciones. En el caso particular del T.P., en el que la cantidad de elementos no nulos de cada fila están dentro de las bandas, y éstas están acotadas por $p + (p + q) = F$, esta complejidad resulta $\mathcal{O}(nF)$. Teóricamente, dado que F no depende de n, podemos decir que la complejidad espacial es lineal en la dimensión de la matriz. Además, en este caso, para puentes grandes (digamos, de más de 25 secciones), mientras la cantidad de ecuaciones es del orden de la centena, F es considerablemente menor (un orden de magnitud, por lo menos), con lo cual efectivamente no es significativo en términos prácticos. En este aspecto, la estructura es notablemente mejor que la solución trivial del arreglo bidimensional, ya que, por un lado, tiene un grado de complejidad menor, y por otro, en ningún momento tiene reservada memoria que no necesite efectivamente.

2.2.2. Complejidad Temporal

En términos generales, sabemos que reemplazar un arreglo por una lista enlazada degrada automáticamente la performance del acceso aleatorio a los elementos. Esto es efectivamente así, dado que para acceder a un elemento cualquiera debemos iterar sobre los elementos de la fila hasta encontrarlo (o alcanzar un elemento con un índice mayor, lo que indica que el elemento buscado no está representado, con lo cual, el valor requerido es 0). Sin embargo, en estas condiciones, acceder al primer elemento no nulo (o a cualquiera de sus anteriores) insume tiempo constante ya que para esto no es necesario iterar por los elementos de la lista, simplemente alcanza con acceder al primero. Veremos que éste es el único acceso "aleatorio" que realiza nuestro algoritmo, con lo cual, no representa un problema para nosotros.

Veamos ahora cómo resultan las operaciones por filas, en las que se basa nuestro algoritmo. Por un lado, para intercambiar dos filas, dado que las representaciones de éstas son completamente independientes en memoria, basta con intercambiarlas en el arreglo de filas de la matriz. Si pensamos que, en general, las listas se representan como un puntero al primer elemento, basta intercambiar los correspondientes a las filas involucradas, lo que insume tiempo constante. (Más allá de que la representación efectiva de las listas utilizadas por la biblioteca estándar que estamos utilizando sea más compleja que un único puntero, en cualquier caso, es una cantidad constante de información, con lo cual el tiempo requerido para realizar el intercambio sigue siendo una cantidad constante.) Veamos ahora la operación que a una fila le suma un múltiplo de otra. Si conseguiéramos realizar esta operación en tiempo lineal, no estaríamos degradando la performance con respecto a la representación con arreglos. Veremos que efectivamente podemos conseguir esto, utilizando una estrategia muy similar al paso *merge* de *Merge-Sort*. El pseudocódigo propuesto es el siguiente:

Algoritmo 1 $F_i \leftarrow F_i + k * F_t$

```
itF_i := iteradorAlInicio(F_i)
itF_t := iteradorAlInicio(F_t)
Mientras itF_i ≠ fin(F_i) e itF_t ≠ fin(F_t)
    Si itF_i → indice < itF_t → indice entonces
        avanzar(itF_i)                    ▷ El próximo elemento no nulo de F_t es posterior al actual no nulo de F_i
    Si no
        Si itF_i → indice = itF_t → indice entonces ▷ Ambas filas tienen un elemento no nulo en la misma posición
            itF_i → valor += itF_t → valor * k
            Si itF_i → valor ≈ 0 entonces
                eliminarActual(itF_i)
            Si no
                avanzar(itF_i)
            Fin Si
            avanzar(itF_t)
        Si no                             ▷ F_t tiene un valor no nulo donde F_i tiene un 0
            F_i.insertarActual(itF_i, *itF_t)
            itF_i → valor *= k
            avanzar(itF_t)
        Fin Si
    Fin Si
Fin Mientras
Mientras itF_t ≠ fin(F_t)
    F_i.agregarAlFinal(*itF_t)            ▷ Todavía quedan elementos para considerar en F_t
    F_i.último.valor *= k
Fin Mientras
```

Observemos que el primer ciclo efectivamente "intercala" ordenadamente los elementos de F_t adecuadamente

multiplicados entre los de F_i en caso de que alguna de las dos filas tenga un elemento nulo en la posición considerada (con lo cual, evita realizar operaciones aritméticas que involucren ceros), y en caso de que ambas filas cuenten con un elemento en la misma posición, almacena la suma correspondiente. Realiza esto mientras existan elementos no nulos en ambas filas (pues cuando éstos se agotan en alguna, se alcanza el final de la fila invalidando la guarda del ciclo, ya que no almacenamos elementos nulos). Si se agotaron los elementos de F_t, el segundo ciclo no se ejecuta, dejando inalterada la última porción de F_i, lo cual es correcto ya que las operaciones que realizaría sería sumar un 0 a las posiciones no nulas del final de F_i. Por el contrario, si restan elementos en F_t (pero no en F_i), simplemente hay que agregar éstos multiplicados por k al final de F_i (ya que correspondería con sumar éstos a ceros de F_i). Notemos dos cosas: por un lado, todas las operaciones que involucran elementos nulos no sólo no se computan, sino que se ignoran por completo, lo cual, para una matriz esparsa, representa una optimización no despreciable; por otro, en ningún momento el algoritmo depende de la estructura en bandas de la matriz, con lo cual funciona para matrices en general, resultando provechoso para matrices esparsas cualesquiera. Veremos que en el algoritmo de eliminación esto nos permite desentendernos de los valores de las bandas de la matriz, para los cuales, durante el transcurso del algoritmo no conocemos sus valores exactos sino cotas, nuevamente evitándonos computar operaciones triviales sin necesidad de predecir qué coeficientes son nulos.

Veamos, finalmente, que este algoritmo tiene la complejidad deseada. Todas las operaciones que realiza las podemos suponer de costo constante ya que involucran avanzar en un paso o desreferenciar los iteradores u operaciones aritméticas. Además, como la condición de corte de los ciclos está dada por alcanzar el final de la listas y nunca retrocedemos los iteradores, podemos afirmar que en el peor caso, entre los dos ciclos, se producen tantas iteraciones como elementos no nulos tuvieran éstas. Con lo cual, realizamos $\mathcal{O}(nZ(F_i) + nZ(F_t))$, que en el caso general puede ser tan malo como $2n$, lo que resulta en la complejidad lineal que buscábamos. Además, en el caso particular del T.P., como vimos en lo referente a la complejidad espacial, este valor está acotado por $F = 2p + q$, que no sólo es constante con respecto a n, sino significativamente menor que éste. Luego, teóricamente podemos afirmar que esta operación tiene costo constante en función de n, y prácticamente, que su costo es un valor menor dentro del algoritmo de eliminación.

2.3. Algoritmo de Resolución

Como dijimos, utilizaremos el método de Eliminación de Gauss con pivoteo parcial para conseguir una matriz triangular superior cuyo sistema asociado sea equivalente al planteado al comienzo. En este paso aprovecharemos el hecho de contar con una matriz banda (tanto al comienzo como en cada paso del método) para acotar la búsqueda del pivote. Básicamente, si estamos en la iteración $j$ésima, necesitamos anular la $j$ésima columna de la matriz, para las filas inferiores a la $j$ésima. Recordando la definición de matriz banda, sabemos que $a_{i,j} = 0$ cuando $j \leq i - p$, o lo que es equivalente, cuando $i \geq j + p$, con lo cual sólo tiene sentido buscar el pivote en las filas $j < i < j + p$. Para el resto del algoritmo, no introdujimos modificaciones. En pseudocódigo:

Algoritmo 2 $triangular(A, b, bandaInferior)$

```
Para j desde 1 hasta n
    ultimaFila := min(n, j + bandaInferior)
    i := buscarPivote(A, j + 1, ultimaFila)
    Si i ≠ j entonces
        A.intercambiarFilas(i, j)
        b.intercambiarCoords(i, j)
    Fin Si
    pivote := A[j, j]
    Para i desde j + 1 hasta ultimaFila
        a_{i,j} := A[i, j]
        Si a_{i,j} ≉ 0 entonces
            multiplicador := a_{i,j}/pivote
            A.sumarMultiploDeFila(i, j, −multiplicador)
            b[i] -= b[j] ∗ multiplicador
        Fin Si
    Fin Para
Fin Para
```

El pseudocódigo expuesto corresponde a la implementación del método de Eliminación de Gauss con pivoteo sin mayores modificaciones. Como dijimos, la única diferencia está en la operación *buscarPivote*, que está restringida a las filas donde no podemos asegurar que no haya coeficientes nulos. La implementación de ésta es una búsqueda trivial de máximo módulo entre los elementos $a_{i,j}$, donde i pertenece al rango explicado anteriormente y j es la columna correspondiente a la iteración.

Analicemos su complejidad. Sabemos que en el caso general, este método tiene complejidad $\mathcal{O}(n^3)$, veamos cómo lo afecta las reducciones aplicadas. Observemos las cotas: por un lado, como vimos antes, $ultimaFila - (j + 1) \leq p <$

$2p+q=F$, con lo cual, tanto el segundo ciclo como la búsqueda del pivote, iteran a lo sumo esta cantidad de veces; el resto de las operaciones involucradas tienen costo constante, a excepción de $sumarMultiploDeFila$, que como vimos, tiene complejidad $\mathcal{O}(F)$. Esto es claro, salvo para los accesos a elementos de la matriz para los que, como dijimos al comienzo, no podemos garantizar siempre su ejecución en tiempo constante. Sin embargo, por cómo funciona el algoritmo, en la iteración $j$ésima en las columnas $j' < j$ ya se ha producido la eliminación, con lo cual, el primer elemento no nulo de las filas consideradas aparece, por lo menos, en la posición $j+1$. En estas condiciones, siempre estamos preguntando por el primer elemento no nulo de la fila (tanto en el valor de *pivote* y $a_{i,j}$, como dentro de $buscarPivote$, lo que nos permite afirmar que, efectivamente, estas operaciones tienen costo constante. Luego, cada iteración realiza las siguientes cantidades de operaciones: $\mathcal{O}(F)$ para buscar el pivote, $\mathcal{O}(F)$ veces suma de dos filas, lo que tiene costo $\mathcal{O}(F)$, y una cantidad constante de otras operaciones también constantes. Luego, el cuerpo del ciclo insume $\mathcal{O}(F^2)$ operaciones, y por lo tanto, todo el algoritmo tiene complejidad $\mathcal{O}(nF^2)$. Como antes, suponiendo F una constante acotada, asintóticamente el algoritmo de eliminación se realiza en tiempo lineal. En la práctica, la relación entre F^2 y n puede resultar más significativa que en los casos anteriores, pero si comparamos nF^2 contra n^3 estaríamos comparando F^2 contra n^2, donde, si F es significativamente menor que n, esto resulta todavía más a favor de F.

Una vez triangulada la matriz despejamos los valores de las variables utilizando el clásico algoritmo de *backward substitution*. Para aprovechar la representación esparasa de la matriz, utilizamos un iterador reverso en cada fila.

Algoritmo 3 $despejar(A, b)$

```
solucion[n] := 0...0
Para i desde n hasta 1
    solucion[i] := b[i]
    itF = iteradorReverso(A.fila(i))
    Mientras itF ≠ principio(A.fila(i)) e itF → indice > i
        solucion[i] -= (itF → valor) * solucion[itF → indice]
        retroceder(itF)
    Fin Mientras
    solucion[i] /= A[i,i]
Fin Para
```

Al igual que en el análisis de los casos anteriores, vemos que todas las operaciones involucradas en el algoritmo tienen costo constante (aún el acceso $A[i,i]$). El ciclo principal se repite n veces, y el ciclo interior itera tantas veces como elementos no nulos tenga la fila luego de su diagonal. Es decir, a lo sumo el ancho de la banda superior, que es $p+q < 2p+q = F$. Luego, el costo de despejar las variables del sistema es $\mathcal{O}(nF)$, para el que, con los mismos argumentos que antes, podemos afirmar que es lineal y eficiente.

2.4. Heurística

El algoritmo anterior permite calcular las fuerzas ejercidas por cada link en el puente, y por ende, la fuerza máxima en módulo. Luego, dada la fuerza máxima que pueden ejercer los links, podemos saber si el puente es o no seguro. Si el puente no es seguro, proponemos un método heurístico para colocar pilares auxiliares en el puente para luego analizar la seguridad del puente original, analizando la seguridad de las dos estructuras generadas por la inserción del pilar. Además de la inserción de pilares, el método propuesto calcular el costo total que demanda estas inserciones.

A continuación, detallamos el algoritmo heurístico:

Algoritmo 4 $calcuarCostoTotal(puente)$

```
costo_estructura, cant_pilares := calcularCostoEstructura(puente)
Devolver costo_estructura * (cant_pilares - 1) * COSTO_PILAR_UNITARIO
```

Algoritmo 5 *calcularCostoEstructura*(*puente*)

```
costo_estructura := 0
cant_pilares := 0
resolverPuente(puente)
Si el puente no es seguro entonces
    Si n > 2 entonces                                   ▷ El puente se puede seguir subdividiendo
        junta := elegirJunta(puente)
        sub_estructura_1, sub_estructura_2 := generarSubEstructuras(junta)
        costo_sub_estructura_1, cant_pilares_1 := calcularCostoEstructura(sub_estructura_1)
        costo_sub_estructura_2, cant_pilares_2 := calcularCostoEstructura(sub_estructura_2)
        costo_estructura+ = costo_sub_estructura_1 + costo_sub_estructura_2
        cant_pilares+ = cant_pilares_1 + cant_pilares_2 + 1
    Si no
        costo_estructura := +∞
        cant_pilares := −1                              ▷ La heurística no puede asegurar el puente.
    Fin Si
Si no
    costo_estructura := puente.calcularCostoParcial()
Fin Si
Devolver costo_estructura, cant_pilares
```

Las clave de esta heurística reside en la elección de la junta, debajo de la cual se colocará un pilar. Optamos por elegir la junta más cercana al link sobre el cual se ejerce la fuerza máxima en módulo. Esta decisión intenta, mediante la inserción de un pilar, relajar la fuerza que se genera alrededor de la junta donde se ejerce la fuerza más grande en módulo. Por otro lado, para elegir la junta tenemos en cuenta que se respete la simetría de las subestructuras generadas. En cuanto a la generación de las subestructuras, sólo hay que calcular el nuevo span para cada una. De esta manera, se van calculando los costos parciales de las subestructuras recursivamente así como también la cantidad de pilares utilizados, para finalmente realizar el cálculo del costo total en base a los costos parciales y la cantidad total de pilares insertados para asegurar la seguridad del puente.

3. Mediciones

3.1. Construcción de las entradas

Para generar los archivos de entrada hicimos un script en lenguaje Python.

El primer bloque de código genera un archivo llamado "medicionesConSpanVariable.in". Se fijan primero las constantes *h*, *carga*, *C* y *fMax* (Las dos últimas no serán utilizadas, se agregan por consistencia, la carga se usará para todas las juntas por igual y será fija, al igual que la altura). Luego se procede a ciclar el span entre un valor mínimo y uno máximo, según el paso establecido, lo mismo se hace para *n*, en un ciclo interno. Como resultado el archivo tendrá varias instancias separadas por líneas en blanco, con el mismo formato que pide la cátedra, con altura y cargas fijas, y span y secciones variables.

El segundo bloque de código hace algo muy parecido, generando el archivo llamado "medicionesConCargaVariable.in". En vez de fijar *carga* se fijará *span*, y se procederá a ciclar entre cargas mínimas y máximas. Éste archivo tendrá varias instancias separadas por líneas en blanco, con altura y span fijo, y cargas y secciones variables.

3.2. Resultados

3.2.1. Span Variable

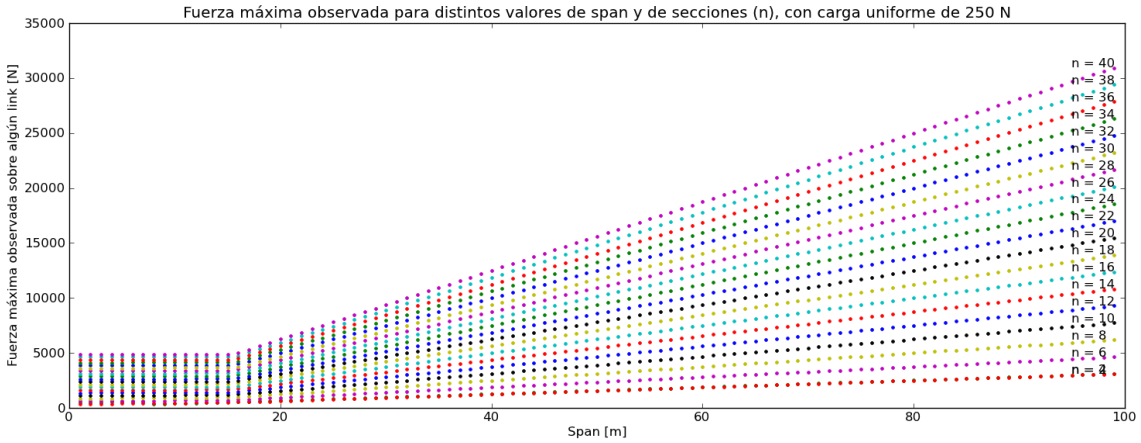


Figura 6: Fuerza máxima ejercida para distinta cantidad de secciones. Variación del valor del span.

Este experimento se realizó con una altura igual a 4m, y una carga igual a 250N en cada junta, variando la cantidad de secciones entre 2 y 40 (siempre considerando los valores pares), y para cada uno de estos, valores de span entre 1 y 100.

Observamos en la figura 6 que luego de una meseta para valores bajos de span, la fuerza máxima crece linealmente tanto con el span como con el valor de n. Estos resultados se corresponden con la intuición: puentes más largos tienen que soportar fuerzas mayores que puentes más cortos, lo mismo que puentes con más secciones, ya que éstos tienen mayor cantidad de cargas, pues colocamos una por junta (esto tiene sentido si pensamos que pueden representar el peso de los tramos de cada sección que agregamos). También observamos que la fuerza máxima crece más rápidamente con el span para valores más grandes de n, lo que también podemos explicar si pensamos en que puentes con más secciones soportan mayor cantidad de carga en total.

Tambien notamos la presencia de la meseta para valores bajos de span (menores a 20m). Presentamos un detalle de esta sección en la figura 7

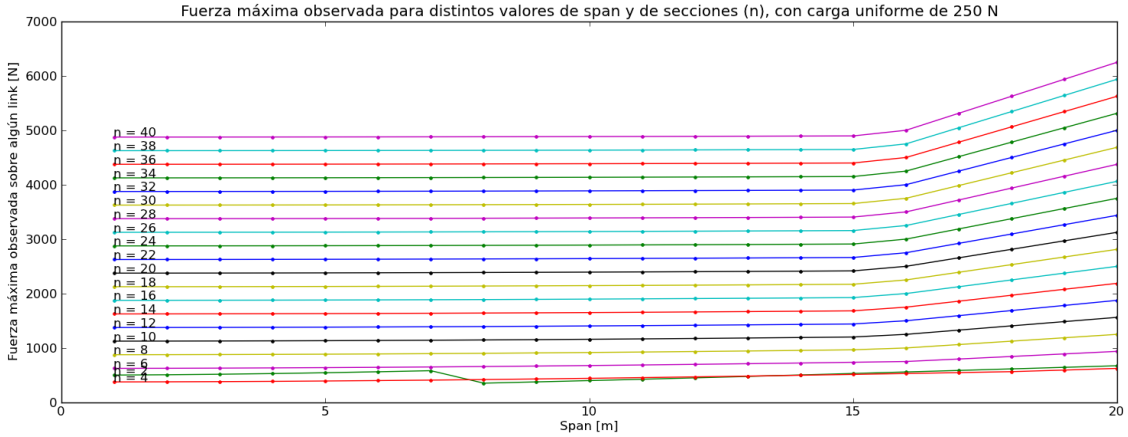


Figura 7: Detalle del gráfico de la figura 6. Variación del valor del span entre 1 y 20 m.

Aquí observamos en la mayoría de los casos que el valor de la fuerza máxima permanece constante para las primeras mediciones, hasta valores de span de, aproximadamente, 15m. Suponemos que para estos valores estamos sobredividiendo el puente, con lo cual las cargas aplicadas no alcanzan para verdaderamente forzar la estructura. Observamos también un comportamiento particular para $n = 2$, que podemos atribuir a la forma particular que adopta el puente en este caso (por ejemplo, no tiene links oblicuos en el sentido contrario a los dos laterales). Más allá de todo esto, en todos los casos se nota con más claridad que, como dijimos antes, para los puentes con más divisiones, la fuerza máxima soportada crece más rápido.

3.2.2. Carga Variable

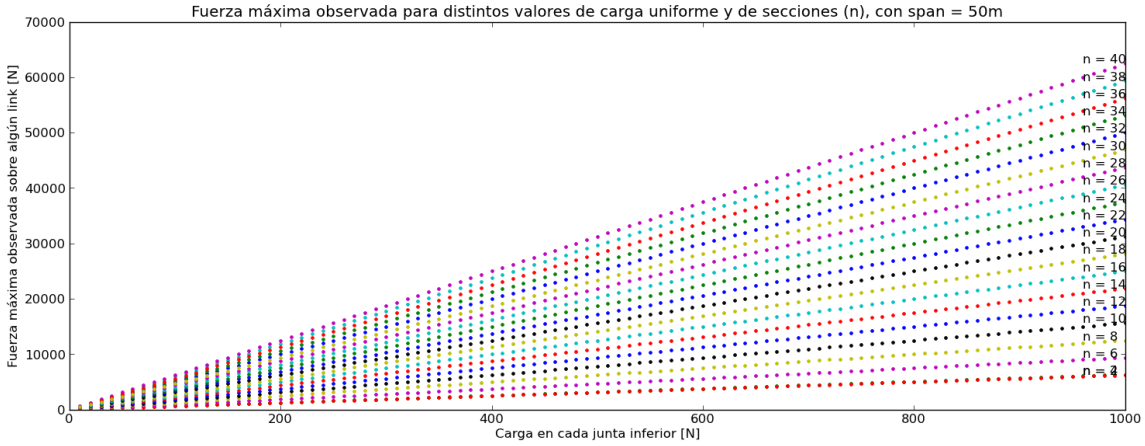


Figura 8: Fuerza máxima ejercida para distinta cantidad de secciones. Variación de la cantidad de carga aplicada sobre cada junta.

Este experimento se realizó con una altura igual a 4m, span igual a 50m y la carga sobre cada junta variando entre 1 y 1000N en pasos de a 10N (siendo siempre misma en todas las juntas). También variamos para cada una de estas situaciones la cantidad de secciones de la misma manera que en el caso anterior ($2 \leq n \leq 40$, con n par). Los resultados se ven en la figura 8

En este caso caben las mismas observaciones que en el caso anterior, más aún, sin la salvedad de la meseta para los valores bajos. Observamos que en estructuras más divididas (y por ende más cargadas) la fuerza soportada crece más rápido, pero siempre lo hace de manera lineal: es decir, una vez fijada la forma de la estructura del puente, la fuera máxima que se ejerce es proporcional a la cantidad de carga aplicada sobre éste.

4. Conclusiones

Para realizar este T.P. resolvimos los siguientes problemas:

- Implementar una estructura de datos para representar eficientemente una matriz esparsa.
  La solución propuesta, representando las filas como listas de elementos no nulos, tiene complejidad espacial $\mathcal{O}(nZ)$ (siendo nZ la cantidad de elementos no nulos almacenados) y permite realizar operaciones de filas (intercambios, suma de un múltiplo de una a otra) en tiempo $\mathcal{O}(n)$, aunque degrada el acceso aleatorio a los elementos a esta misma complejidad. Sin embargo, vimos que esto no es un problema para el uso que le dimos en nuestra solución.

- Construir un sistema lineal genérico para la estructura de puentes Pratt-Truss.
  Numerando tanto los links del puente como las juntas de manera adecuada, en forma genérica dependiendo de n (la cantidad de secciones del puente, que siempre consideramos par), pudimos construir el sistema de ecuaciones de las fuerzas sobre las juntas para cualquier puente Pratt-Truss conociendo algunos parámetros básicos (cantidad de secciones, largo y altura del puente, y cargas aplicadas). Vimos además, que esta numeración produce un sistema cuya matriz asociada tiene estructura de bandas 4, 6.

- Resolver un sistema lineal utilizando el método de Eliminación de Gauss.
  Para esto realizamos una implementación bastante directa del método con pivoteo parcial que optimizamos aprovechando la información sobre la estructura de bandas de la matriz del sistema. Además, dada la estructura de datos que utilizamos, conseguimos un algoritmo con complejidad temporal $\mathcal{O}(n)$.

- Implementar un algoritmo heurístico para cubrir una determinada distancia con estructuras de puentes Pratt Truss, considerando como limitante de la estructura la fuerza máxima que se ejerce sobre algún link.
  Para esto, considerando el costo de insertar pilares de concreto bajo alguna(s) junta(s) del puente, implementamos una heurística recursiva que subdivide las estructuras y recalcula las fuerzas ejercidas hasta que ninguna sobrepase determinado límite.

Con todo esto conseguimos, por un lado, calcular las fuerzas que soportan los links de una estructura Pratt-Truss en las condiciones dadas, y por otro, utilizar este cálculo para encontrar una serie de subestructuras de este tipo que puedan reemplazar a una más grande que no es considerada segura, intentando minimizar el costo de construcción.

Además, experimentamos utilizando este modelo para observar cómo aumentaban las fuerzas que se ejercían sobre los links al modificarse ciertas variables estructurales. Vimos que, en general, la fuerza máxima es proporcional a la

carga soportada (cuando ésta se aplica uniformemente sobre las juntas) y que ésta crece más rápido para puentes más subdivididos (y por ende, más cargados, si consideramos dentro de la carga el peso de la estructura propia). Al variar la longitud del puente, observamos que para valores bajos (de hasta 15m, aproximadamente), la fuerza máxima ejercida no varía, lo que nos dio la pauta de que en esa situación la estructura trabaja sin realizar mayores esfuerzos. En cambio, para valores más grandes observamos lo mismo que en el caso anterior: crecimiento proporcional y más pronunciado para los puentes más subdivididos.