Título: ***Market Risk Engine***
Volumen: ***Único***
Alumno: ***Xavier Guirao Soler***
Director: ***Oliver García Ranea***
Departamento: ***Empresa***
Fecha: ***13 junio de 2011***

DATOS DEL PROYECTO

Título del proyecto: Market Risk Engine

Nombre del estudiante: Xavier Guirao Soler

Titulación: Ingeniería técnica informática de gestión

Créditos: 22,5

Director: Oliver García Ranea

Departamento: Empresa

MIEMBROS DEL TRIBUNAL

Presidente: Joaquim Deulofeu Aymar

Vocal: Marta Jiménez Castells

Secretario: Joan Carles Gil Martin

CALIFICACIÓN

Calificación numérica:

Calificación descriptiva:

Fecha:

Dedicado a mis padres y a mi familia por toda la ayuda y apoyo recibido durante todos estos años…

… y en particular a mi abuelo, hasta siempre.

Índice

1. Introducción 10
2. Visión global del proyecto 11
2.1. Contexto actual 11
2.2. Finalidad del proyecto 11
2.3. Objetivos 12
2.4. Organización de la documentación 13
2.5. Metodología de trabajo 14
2.6. Entorno de trabajo 15
2.7. Fases del desarrollo del proyecto 15
2.8. Análisis de requisitos 17
2.8.1. Requisitos funcionales 17
2.8.2. Requisitos no funcionales 17
3. Conceptos básicos 18
3.1. ¿Qué es una Security? 18
3.2. ¿Qué es un Portfolio? 20
3.3. ¿Qué es un Benchmark? 20
3.4. ¿Qué es una posición? 21
3.5. ¿Qué es el Market Risk? 21
3.6. ¿Qué diferentes riesgos componen el Market Risk? 21
3.6.1. Inputs 21
3.6.2. Outputs 22
4. Planificación 27
4.1. Planificación temporal 27
4.1.1. Tabla de tiempos 27
4.1.2. Desglose de tareas 27
4.1.3. Diagramas de Gantt 29
4.1.4. Desviaciones temporales 31
4.2. Planificación económica 32
4.2.1. Contexto actual 32

4.2.2. Trabajador autónomo 33
5. Benchmarking 35
5.1. Análisis aislado 37
5.1.1. Tiempo de respuesta 37
5.1.2. Uso de la CPU 38
5.1.3. Uso de la memoria 42
5.1.4. Visión global 49
5.2. Análisis multithread 50
5.2.1. Tiempo de respuesta 50
5.2.2. Uso de la CPU 51
5.2.3. Uso de la memoria 55
5.2.4. Visión global 61
5.3. Conclusión 62
5.3.1. Thread 62
5.3.2. Multi-thread 63
5.3.3. Conclusión final 64
6. Diseño del sistema 65
6.1. Clases del diseño 65
6.1.1. Programa de captura de datos 65
6.1.2. Programa testeo de tiempos 68
6.1.3. Librería 73
6.2. Diagrama del sistema 80
6.2.1. Módulos 81
6.2.2. Comunicación entre los módulos 82
6.3. Casos de uso 84
6.4. Testeando la librería 88
6.5. Uso de la librería en un proyecto externo 89
6.6. Inclusión en el modelo PAT One 90
7. Conclusión 91
7.1. A nivel de aplicación 91
7.2. A nivel personal 91

7.3. Problemas encontrados 91
7.4. Futuras mejoras 92
8. Bibliografía 93
8.1. Libros y *papers* 93
8.2. Páginas web 93
9. Anexos 95
9.1. Tabla de procesos en ejecución 95
9.2. Tabla de consumo CPU 96
9.3. Tabla de consumo de memoria RAM 97
9.4. Herramientas extras para realizar el proyecto 98

1. Introducción

OTC Fin es una consultoría tecnológica orientada a empresas del sector de la banca privada (Bancos de Inversión, Aseguradoras, Bancos, *Hedge Funds…*). Trabajando en relación directa con los departamentos de riesgo financiero de nuestros clientes, les ofrecemos herramientas de integración de datos, *reporting*, análisis de riesgo, *p&l* y *attribution*.

Como parte de la nueva estrategia adoptaba por OTC Fin en el último año, y basándose en la experiencia adquirida desde 1990, se está desarrollando la plataforma *PAT One*. Esta plataforma fusiona los mundos del *Asset Management* con el *Management* de *Fund of Funds* y *Hedge Funds*. De esta manera los usuarios pueden tener desde un control y visión de la primera transacción de una opción hasta la estrategia de inversión del *Hedge Fund.*

PAT One es una plataforma de análisis de riesgo que consta de los siguientes módulos:

- *PAT One KnowVault*: *Datawarehouse* que combina los mundos de *Fund of Funds* con *Asset Management*. Capaz de mantener datos diarios de más de un millón de posiciones con un gran rendimiento.

- *PAT One Harvester*: Diseñado completamente con tecnología Microsoft. Es el encargado de hacer las funciones ETL de extracción, transformación, integración y carga de los diferentes proveedores de datos.

- *PAT One Engines*: Los *engines* son los encargados de hacer los cálculos financieros dando un valor añadido extra a la integración de datos en *PAT One KnowVault*. Son piezas independientes que realizan su tarea coordinadas por el *Workflow Engine*. Es aquí donde este proyecto tiene su objetivo ya que encargará de desarrollar el *Market Risk Engine*.

- *PAT One Reporter*: El encargado de mostrar al usuario los datos extraídos de *KnowValut* y los *Engines*. Con capacidad de generar *What-if* scenarios, programar *reports* y exportar a formatos estandards.

2. Visión global del proyecto

El contexto actual del proyecto así como la finalidad, los objetivos, el entorno de trabajo y restricciones, las necesidades para el usuario y los análisis de requisitos estarán presentes en este apartado.

2.1. Contexto actual

Como se ha expuesto en la introducción, OTC Fin dispone de una estructura llamada *PAT One* que da una serie de servicios a varias empresas de ámbito económico. Con el objetivo de ofrecer nuevas herramientas, se ha decidido crear una librería que permita obtener de manera rápida todos los valores del Riesgo de Mercado.

El módulo donde estará esta librería es el del *Engine*. Contiene tres zonas claramente diferencias:

- *Analytics.*
- *Aggregation grouping.*
- *Risk.*

La última es donde se integrará esta librería.

2.2. Finalidad del proyecto

La finalidad del proyecto es la creación de una librería en C# que permita el cálculo de todos los Riesgos de Mercado. Esta librería ha de tener las siguientes características.

- Rápida.
- Poco uso de CPU.
- Uso mínimo memoria.
- Independiente.
- Fácil uso.

La primera y esencial es que sea rápida. La librería será llamada con mucha frecuencia, por tanto es de esperar que se realicen todos los cálculos en el menos tiempo posible.

La segunda y la tercera característica están muy ligadas. Debido a la gran cantidad de registros que se usarán para obtener los resultados finales, se ha de codificar una librería que gestione de la mejor forma posible tanto el uso como el consumo de la CPU.

La cuarta es que sea independiente. Si bien se usará en el motor de *PAT One Engine*, la librería ha de ser exportable a otros proyectos futuros que se puedan realizar en la Compañía.

Por último la librería ha de ser de fácil uso comentando el código y nombrando a las variables y métodos de forma que indiquen exactamente qué función hacen.

2.3. Objetivos

Los objetivos del proyecto son:

- Realizar el cálculo de todos los riesgos que forman parte del *Market Risk*. Estos cálculos incluyen los siguientes términos:
    - *Factor Risk*
    - *Total Risk*
    - *VaR (Value at Risk)*
    - *Marginal Contribution to Total Risk*
    - *Marginal Contribution to Specific Risk*
    - *Marginal Contribution to Factor Risk*
    - *Marginal Contribution to VaR*
    - *Contribution to Total Risk*
    - *Contribution to Factor Risk*
    - *Contribution to VaR*
    - *% Total Risk*
    - *% Specific Risk*

- *% Factor Risk*
- *Implied Alpha*
- *Active Exposure Vector*
- *Market Risk beta*

- Documentar de una manera clara el código del programa para facilitar la compresión del mismo.

2.4. Organización de la documentación

La documentación se organizará en los siguientes apartados:

- Introducción: Se realiza una definición de la empresa en la que se desarrollará el proyecto y una situación general de donde se ubicará dentro de la Compañía.

- Visión global del proyecto: Se hará hincapié en lo necesario para entender el por qué del proyecto, así como el entorno de trabajo en el que se realizará, los análisis de requisitos y las necesidades del usuario.

- Planificación: Apartado en el que se explicará cual será la planificación económico-temporal del proyecto.

- Conceptos básicos: Se redactarán todos los conceptos básicos que forman el proyecto. Debido a la falta de conocimientos económicos adquiridos durante la carrera, este apartado es de vital importancia.

- *Benchmarking*: Estudio de librerías que permitan hacer los cálculos atendiendo a los requisitos tanto funcionales como no funcionales.

- Diseño del sistema: La representación detallada del sistema *PAT One*, el diseño de las clases, los diferentes casos de uso, los *tests*, la inclusión de la librería en el modelo *PAT One* y un manual de cómo incluir la librería en un proyecto en Microsoft Visual Studio 2010 estarán presentes en apartado.

- Conclusiones: Punto en el que se expondrán las conclusiones que tanto a nivel personal como a nivel educativo han surgido una vez finalizado el proyecto.

- Bibliografía: Contendrá todos los libros y *papers* así como los recursos web que se han consultado.

- Anexo: Contendrá información adicional del proyecto.

2.5. Metodología de trabajo

De metodologías de trabajo a seguir hay dos, ágiles y pesadas. En la tabla 1 se resumen:

	ÁGILES	PESADAS
Actitud ante los cambios	Adaptación al cambio	Soportar el cambio
Documentación	Poca	Mucha
Objetivo	Valor rápido	Buenas garantías
Ámbito	Sujeto a cambios , desconocido e incierto	Bien conocido, muy estable
Recursos	Presupuesto incierto Económicamente ajustado	Presupuesto suficiente
Tiempo	Sin plazos definidos	Claro y con plazos definidos
Riesgos	Riesgos desconocidos Mayor impacto de tecnología nueva	Riesgos previstos e impacto menor
Arquitectura	Diseñada para las necesidades del momento	Diseñada para las necesidades del momento y del futuro
Desarrolladores	Ágiles y colaborativos	Orientados a procesos con las habilidades adecuadas
Clientes	Colaborativos, dedicados e informados	Informados, representativos y colaborativos
Coste de cambios	Barato	Caro

Tabla 1

Para la elaboración del proyecto se ha decantado hacía una metodología ágil, en concreto la *SCRUM*. Ésta se caracteriza por la inclusión de dos roles:

- Role de cerdo: Son los que están comprometidos con el proyecto y en definitiva los encargados que tire adelante.

- Role de gallina: Es el destinatario del proyecto. Estos usuarios no forman parte del proceso *Scrum* pero si se han de tener en cuenta de cara al desarrollo final.

A grandes rasgos se podría definir esta metodología de trabajo en los siguientes puntos:

Se planean unas reuniones diarias para ver los avances en el proyecto, denominadas *Daily Scrum*. Éstas tienen como finalidad contestar a tres preguntas: ¿Qué se ha hecho desde ayer? ¿Qué se tiene planeado

hacer hoy? y finalmente analizar si se ha tenido algún problema reciente que haya impedido alcanzar alguno de los objetivos propuestos.

Una vez finalizado el *Daily Scrum* se pasará a hacer el *Scrum* de *Scrum*.

Cada 15 días se realiza una reunión de planificación del *sprint* para ver hasta cuanto se puede llegar en los próximos días. Aquí se prepara el *Sprint Backlog* donde se detalla cuánto tiempo se tomará en hacer el trabajo.

Finalmente la fase de *Sprint Retrospective* donde después de cada *sprint* se debatirá que se ha hecho, de qué forma y si hay algo a mejorar.

Como las dimensiones del proyecto no son grandes, se ha hecho una síntesis de esta metodología ya que se espera que la elaboración del proyecto no sea superior a 4 meses como así refleja la planificación prevista[1]. Así pues las reuniones de planificación, en lugar de ser quincenales pasan a ser semanales.

Todo lo anterior se complementará con la herramienta *Visual Studio Team System* para facilitar el seguimiento de la planificación.

2.6. Entorno de trabajo

Como entorno de trabajo en el que se ha realizado el proyecto se encuentra el siguiente *software:*

- Netbeans IDE: Entorno de desarrollo para programadores que cuenta con diversos *plugins* que facilitan la programación en diferentes lenguajes.
- Microsoft Visual Studio 2010: Entorno de Microsoft usado para hacer proyectos de diferente índole.
- Windows 7 Professional de 64 bits: Sistema operativo donde se realizará el proyecto.

2.7. Fases del desarrollo del proyecto

Los siguientes pasos marcarán el ciclo de vida del proyecto. Cada punto de esta lista exigirá un objetivo a conseguir y por tanto el desarrollador junto con el director del proyecto, deberán definir las pautas de cada uno de los *sprints*:

[1] Ver figura 5 página 28

- Familiarización personal con la terminología de análisis de riesgo financiero. El resultado de este análisis servirá para conocer nuevos términos así como para saber las fórmulas que permitan el cálculo de cada uno de los objetivos.

- Selección de la librería. Sabiendo los requisitos no funcionales, se ha de escoger una librería que permitan su cumplimiento. En el estudio constarán los siguientes datos:

    - Tiempo total de cálculo
    - Memoria RAM utilizada
    - Ciclos de CPU y número de procesadores usados
    - % CPU usado
    - Calidad de los datos resultantes
    - *Feedback* del mundo de desarrolladores

  Además se hará un estudio comparativo para cada una de las librerías en diferentes casos de *tests*:

    - *Normal Cases*: Casos semejantes que se darán en la vida útil del proyecto (80%).
    - *Extrem Cases:* Casos extremos donde haya una gran cantidad de registros (10%).
    - *Light Cases:* Casos básicos donde haya una pequeña cantidad de registros (10%).

- Ejecución de los casos de *test*. Se correrá el proyecto en cada uno de los casos definidos anteriormente para saber que librería es la mejor entre las candidatas.

- Diseño, implementación. Se codificará la aplicación con la librería ganadora del *Benchmarking*.

- Testeo de la aplicación. Se compararán los resultados obtenidos de la librería con los que ya dispone la Compañía en los reportes de la herramienta *PAT One Reporter*.

- Integración de la librería en *PAT One*. Parte final del proyecto donde se integrará la librería en el marco global del sistema *PAT One*.

2.8. Análisis de requisitos

De requisitos del *software* se pueden encontrar de dos tipos: Los funcionales y los no funcionales. Los funcionales definen el comportamiento del sistema así como las funcionalidades que éste ha de cumplir mientras que los no funcionales establecen las necesidades básicas que ha de tener el sistema a nivel de implementación y diseño.

A continuación se expondrán que requisitos ha de cumplir la aplicación.

2.8.1. Requisitos funcionales

El *Engine* de *PAT One* ha de saber calcular todos los riesgos que se han descrito en los objetivos así como obtener el *Active Vector* asociado y la Beta del *Market Risk*.

2.8.2. Requisitos no funcionales

Se encuentran los siguientes:

- Debido a la utilización del *Engine* en modo *What-if*[2], el tiempo de respuesta ha de ser muy corto.
- La posibilidad que el *Market Risk Engine* pueda estar corriendo paralelamente (*Aggregation*, *p&l*, etc.) a otros motores obliga a minimizar el uso de los recursos de la CPU.
- El *Engine* debe ser totalmente independiente, y reusable en cualquier parte del *software PATOne*. No ha de hacer ninguna llamada a la base de datos, ni a ficheros, ni a formatos específicos.
- El proyecto tiene que estar desarrollado en lenguaje Microsoft .NET C#[3].

[2] Simulación de un estado para ver cómo se comporta la *Security* con determinados factores. ¿Qué pasaría si…?
[3] http://es.wikipedia.org/wiki/C_Sharp

3. Conceptos básicos

Importante apartado donde se pretende explicar todos los conceptos que aparecerán en este proyecto. Será necesario saber contestar a las siguientes preguntas antes de desarrollar la librería:

- ¿Qué es una *Security*?
- ¿Qué es un *Portfolio*?
- ¿Qué es un *Benchmark*?
- ¿Qué es una posición?
- ¿Qué es el *Market Risk*?
- ¿Qué diferentes riesgos componen el *Market Risk*?

3.1. ¿Qué es una Security?

Un valor (*Security)* representa una serie de derechos parciales de un propietario sobre ciertas acciones o un título de crédito u obligación, con características y derechos estandarizados.

Tiene las siguientes características:

- Formulismo: se deben cumplir los requisitos que señala la ley, la falta de uno de éstos puede causar la ineficacia del título. En cada país hay un organismo que se dedica a controlar que estos requisitos se cumplan, así por ejemplo encontramos el caso de España donde se encarga la Comisión Nacional del Mercado de Valores (CNMV) o en Estados Unidos con la *Securities and Exchange Commission (SEC)*.
- Incorporación: quien posee un título de una *Security*, es dueño del derecho contenido en él.
- Legitimación: la persona queda legalmente facultada, por el solo hecho de la tenencia, a ejercer el derecho incorporado en el título (títulos al portador); o por constar como titular del derecho incorporado al título (títulos nominativos).
- Literalidad: determina el alcance y modalidad de los derechos y obligaciones consignadas a él.
- Autonomía: el poseedor ejercita un derecho propio que no puede ser restringido o destruido.

Además de las características anteriores, existen cinco requisitos básicos que toda *Security* debe tener para que se considere válida:

- El nombre del título valor.
- La fecha y el lugar de creación.
- Las prestaciones y derechos que el titulo concede.
- El lugar de cumplimiento o ejercicio de las mismas.
- La firma de quien las da.

Como información complementaria en la figura 1 se puede encontrar la clasificación de las *Securities:*

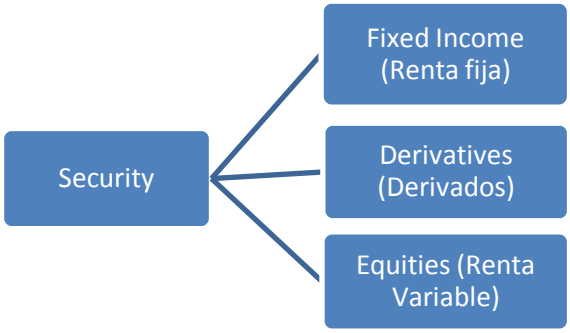

Figura 1

- *Fixed Income*: Son aquellas en las que empresas o gobiernos emiten deuda con el propósito de obtener financiación para que los inversores obtengan un interés continuo e invariable. En el tiempo de la contratación, se conocen tanto las condiciones económicas como las temporales.
- *Derivatives:* Sirven para cubrir el riesgo. El inversor las usa para adelantarse y prevenirse económicamente ante posibles cambios futuros que puedan perjudicar a los activos que posee.
- *Equities:* Aquellos valores en los que el flujo de futuros es incierto, por tanto el inversor no puede conocer de antemano el rendimiento que va a conseguir. Un ejemplo de valores de Renta Variable son las acciones bursátiles.

3.2. ¿Qué es un Portfolio?

A la hora de definir un *Portfolio* (Cartera de Valores) se ha de tener en cuenta la definición de *Security.* Una Cartera de Valores no es más que un conjunto de *Securities* de las cuales el inversor tiene comprados una cantidad de derechos (posiciones). Figura 2.

	Sec. Id	Sec. Name	Act. SMR	Act. VAR	Sec. Type	MV Gross Base	MV Wgt.	CMFR
Total Fund						2,978,902,170	100.000%	16.57
GS CALTEX CORP 5.5 04/24/2017	EG3788221	GS CALTEX CORP 5.5 04/24/2017			CO	8,936,721	.300%	15.22
GLOBO COMUNICACAO E PARTI 7.25 04/26/2022	EG3888575	GLOBO COMUNICACAO E PARTI 7.25 04/26...			CO	20,965,083	.704%	41.25
TNK-BP FINANCE SA 6.625 03/20/2017	EG2641132	TNK-BP FINANCE SA 6.625 03/20/2017			CO	6,991,250	.235%	33.80
ALFA DIVERSIFIED PAYMENT 4.81875 03/15/2012	EG3010089	ALFA DIVERSIFIED PAYMENT 4.81875 03/1...			AB	1,800,253	.060%	10.92
WOORI BANK 6.208 05/02/2037	EG4020624	WOORI BANK 6.208 05/02/2037			CO	5,183,022	.174%	11.87
PETROLEUM CO OF TRINIDAD 6 05/08/2022	EG4211231	PETROLEUM CO OF TRINIDAD 6 05/08/2022			CO	14,074,514	.472%	41.61
INTERGAS FINANCE BV 6.375 05/14/17	EG4263059	INTERGAS FINANCE BV 6.375 05/14/17			CO	9,255,000	.311%	34.70
PROVINCE OF BUENOS AIRES 9.375 09/14/2018	EF7962626	PROVINCE OF BUENOS AIRES 9.375 09/14...			GB	277,500	.009%	22.83

Figura 2

Así pues una cartera de valores puede estar formada por varios tipos de *Securities*. Hay estrategias más arriesgadas, donde todas las *Securities* son activos de renta variables (ver figura 3, Cartera de Valores 2) u otras de menos (ver figura 3, como Cartera de Valores 1 y 3) donde se mezclan varios tipos.

Cartera de Valores 1	Cartera de Valores 2	Cartera de Valores 3
•Activo Renta Fija	•Activo Renta Variable	•Derivado
•Derivado	•Activo Renta Variable	•Activo Renta Fija
•Activo Renta Variable	•Activo Renta Variable	•Derivado
•Activo Renta fija	•Activo Renta Variable	•Activo Renta Variable

Figura 3

3.3. ¿Qué es un Benchmark?

Benchmark son aquellas carteras estandards publicadas por los bancos de inversión y que definen el comportamiento del mercado de una o varias áreas como:

- Deuda pública.
- Deuda corporativa.
- Alto riesgo.

3.4. ¿Qué es una posición?

Se define por posición a7quella *Security* asociada a un *Portfolio* adquirida por un *Portfolio Manager* en una fecha en concreto.

3.5. ¿Qué es el Market Risk?

Market Risk (Riesgo de Mercado) es el conjunto de pérdidas posibles que se producen en un escenario concreto de mercado.

¿Cómo se hace para calcular el Riesgo de Mercado?

Lo primordial es mirar que factores de riesgo influyen en nuestra Cartera de Valores. Los factores de riesgo son todas las variables, tanto económicas como sociales y demográficas que pueden afectar a un *Portfolio*. Una vez se dispone de los factores, se analizan sus tendencias mediante una recolección histórica. A continuación se hace un análisis de sensibilidad que permita conocer con exactitud cómo se comportan los diferentes activos ante los factores anteriores. Esta metodología es conocida como VaR.[4]

3.6. ¿Qué diferentes riesgos componen el Market Risk?

En el *Market Risk* existen varios tipos de riesgo. A continuación se clasificarán en dos grupos en función de si son parámetros de entrada (*inputs*) o bien si son de salida (*outputs*).

3.6.1. Inputs

Se dispone de los siguientes datos de entrada para hacer los cálculos de la librería:

- Double *Specific Risk.* Riesgo que afecta a una pequeña cantidad de activos.
- Double *Security Weight*. Peso de la *Security* para un *Portfolio* concreto.
- Double *Security Specific Risk*. Riesgo especifico de una *Security* dentro de una cartera de valores.
- Matrix *Covariance*. Matriz que contiene la covarianza de los elementos de un Vector *Security Exposure Vector.* Parte de la *Security* que está expuesta al riesgo de un área concreto.

[4] Ver página 23

- Vector *Portfolio Exposure Vector*. Parte del *Porfolio* que está expuesto al riesgo de un área concreto.

- Vector *Active Exposure Vector.* La diferencia entre el *Portfolio Exposure Vector* y el *Benchmark Exposure Vector*. También se puede encontrar como un posible *output*.

- Int *Number of Business days.* Días laborales al año.

- Int nDays. Días en los que se calcula el VaR.

- Double *Nivel de confianza*. Valor utilizado para el cálculo del VaR.

3.6.2. Outputs

El proyecto constará de cuatro partes claramente diferenciadas:

- Calculo de las métricas que componen el *Market Risk*

- Obtención de los marginales de cada uno de los riesgos anteriores.

- *Beta* del *Market Risk*.

- Generación del *Active Exposure Vector*.

3.6.2.1. Market Risk Metrics

- *Factor Risk*: Riesgo de la cartera que describe la relación entre los riesgos y la rentabilidad para los inversores. Se calcula de la siguiente manera:

$$\sqrt{Exposure\ Vector * \ Covariance\ Matrix * Exposure\ Trasposed\ Vector}$$

- *Total Risk:* Suma del riesgo específico y el riesgo del factor en una cartera o para una *Securitie*:

$$\sqrt{Factor\ Risk^2 + Specific\ Risk^2}$$

- *Value at Risk* (VaR): Metodología para medir el riesgo de mercado de un activo o cartera de valores de activos financieros, es decir cuantifica la pérdida máxima que una cartera puede tener. Hay un término en el operando del cálculo del VaR que será básico, éste es el nivel de confianza. Se define como el porcentaje de acierto que puede tener el cálculo del VaR.

Por tanto si se dispone de un nivel de confianza del 95%, se puede afirmar que con un 95% de posibilidades la pérdida máxima será aquella que haya resultado del cálculo del VaR. En la figura 4 se muestra una distribución normal posible generada.

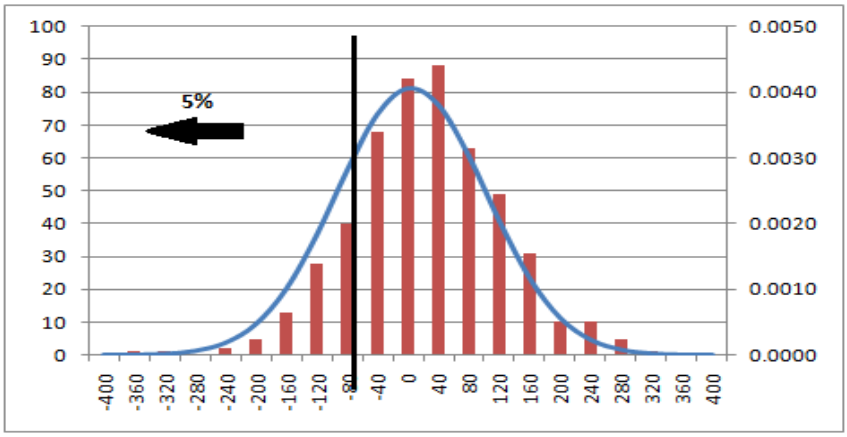

Figura 4

Los números negativos cuantifican pérdidas, mientras que los positivos ganancias. Hay una línea que separa el gráfico del resto con un limitador del 5%, esto es un concepto alternativo al VaR y es la Pérdida Esperada en la Cola (ETL). Existe un 5% de posibilidades que se produzcan pérdidas superiores a los 80 puntos, mientras que un 95% que se produzcan pérdidas inferiores a 80 puntos y unas ganancias que en contados cosas pueden llegar a los 320.

Existen dos formas de hacer el cálculo del VaR:

- Metodología de simulación. Se pueden encontrar dos:

  a) Simulación histórica: En función de los rendimientos históricos de los precios de los activos.

  b) Simulación de Montecarlo. En función de la simulación de rendimientos mediante números aleatorios.

- Metodología paramétrica. Basada en las varianzas y covarianzas de los rendimientos de los precios de los activos. Esta será la metodología que se usará en nuestro proyecto para realizar el cálculo del VaR. La fórmula se describe a continuación:

$$Total\ Risk * Inverse\ Normal\ Distribution\ (confiance\ level) * \sqrt{\frac{nDays}{Number\ of\ Business\ Days}}$$

3.6.2.2. Marginals and Contribution to Market Risk Metrics

Por *Contribution* se entiende la contribución de la posición al riesgo total del *Portfolio* para cada una de las métricas anteriores.

- *Contribution to Total Risk (CTR)*:

$$100 * MCTR * Security\ Weight$$

- *Contribution to Factor Risk (CFR)*:

$$100 * MCFR * Security\ Weight$$

- *Contribution to VaR (CVaR):*

$$CTR * Inverse\ normal\ distribution\ (Confiance\ Level) * \sqrt{nDays/Number\ of\ Business\ Days}$$

Se define *Marginal* como el instrumento metodológico que se centra en el análisis de los cambios muy pequeños en las cantidades totales de las variables económicas.

- *Marginal Contribution to Total Risk (MCTR)*:

$$\frac{Sec\ Weight * \ ((\ Specific\ Risk)^{\ 2} + (Covariance\ Matrix * Portfolio\ Exposu\ \ Vector * Security\ ExposuVector\ Transpose))}{100 * Total\ Risk}$$

- *Marginal Contribution to Specific Risk (MCSR)*:

$$\frac{Porfolio\ Exposure\ Vector * Security\ Exposure\ Vector\ Transpose}{100 * \ Portfolio\ Specific\ Risk}$$

- *Marginal Contribution to Factor Risk (MCFR):*

$$\frac{Covariance\ Matrix * portfolio\ Exposure\ Vector * Security\ Exposure\ Vector\ transpose}{100 * Factor\ Risk}$$

- *Marginal Contribution to VaR (MCVaR)*:

$$MCTR \ * \ Inverse\ Normal\ Distribution\ (confiance\ level) * \sqrt{\frac{nDays}{Number\ of\ Business\ Days}}$$

Percent (%) es el % que influye en la variable asociada al *Portfolio,* la *Security* que se está tratando.

- *% Total Risk:*

$$CTR\ /\ Total\ Risk$$

- *% Specific Risk:*

$$(100\ *\ (MCSR\ *\ Security\ Specific\ Risk/100))\ /\ Portfolio\ Specific\ Risk\ /100$$

- *% Factor Risk:*

$$(100\ *\ (MCSR\ *\ Security\ Factor\ Risk/100))\ /\ 100*Factor\ Risk$$

Implied Alpha: Exceso en la rentabilidad esperada de un *stock* para justificar el peso del *Portfolio*.

$$1.5\ *MCTR\ *\ Total\ Risk$$

3.6.2.3. Beta

Cantidad de riesgo respecto del *Benchmark*. A mayor beta, mayor es la diferencia entre el riesgo de la cartera y el riesgo del *Benchmark*. La siguiente fórmula calcula la Beta del *Market Risk*:

$$\frac{Portfolio\ Total\ Risk^2\ *\ Benchmark\ Total\ Risk^2 * \ Active\ Total\ Risk^2}{2 * \ (Benchmark\ Total\ Risk)^2}$$

3.6.2.4. Active Risk or Tracking Error

3.6.2.4.1. Active Exposure Vector

Es el vector de exposición de factores de riesgo resultante de la diferencia entre el vector del *Portfolio* y el vector del *Benchmark*.

3.6.2.4.2. Tracking Error

Tracking Error es la diferencia entre los valores de riesgo de la cartera de inversión y los valores de riesgo del *Benchmark*.

4. Planificación

En todo proyecto es importante saber que se va hacer y cómo va hacer. A continuación se cuantificará tanto el número de horas a trabajar como el coste total del proyecto.

4.1. Planificación temporal

La jornada laboral es de 6 horas diarias que se comprenden entre las 8:00 hasta las 14:00 de lunes a viernes. Se harán dos planificaciones, una antes de iniciar el proyecto y otra una vez acabado. Finalmente se estudiarán las desviaciones temporales aparecidas.

4.1.1. Tabla de tiempos

	Duración prevista	Duración final
Proyecto	576 horas	450 horas
Inicio del proyecto	0 horas	0 horas
Visión global del proyecto	30 horas	30 horas
Estudio de conceptos	30 horas	40 horas
Planificación	12 horas	12 horas
Benchmarking	180 horas	150 horas
Búsqueda de librerías	12 horas	23 horas
Implementación	120 horas	79 horas
Testing	30 horas	42 horas
Conclusiones	18 horas	6 horas
Implementación	186 horas	65 horas
Testing	90 horas	120 horas
Memoria y reuniones	576 horas	450 horas
Final del proyecto	0 horas	0 horas

Tabla 2

4.1.2. Desglose de tareas

El proyecto constará de las siguientes tareas:

- Inicio del proyecto: Se inicia el día 1 de marzo del 2011.

- Visión global del proyecto: Redactado que se explica en el apartado 2. También se harán reuniones con el director del proyecto.

- Estudio de conceptos: Parte básica donde se estudiará toda la teoría financiera que se usará en el proyecto.

- Planificación: Definir el coste tanto temporal como económico que tendrá el proyecto.

- *Benchmarking*: Fase clave del proyecto donde se decidirá que librería se utilizará. Está formado por los siguientes puntos:

    - Búsqueda de librerías: Búsqueda exhaustiva de librerías en lenguaje C# que permitan hacer cálculos de algebra lineal.

    - Implementación: Codificación del *Factor Risk.* Se medirá el tiempo que tarda cada una de las librerías. Finalmente se hará el código Java que permitirá hacer el volcado a un fichero .csv de los datos recogidos para posteriormente hacer los gráficos de medición.

    - *Testing*: Se harán los gráficos, así como un estudio comparativo de todas las librerías.

    - Conclusiones: A partir del estudio anterior, se llegará a una conclusión sobre que librería usar.

- Implementación: Se hará el código de la librería FinLib con la librería ganadora.

- *Testing*: Se compararán los valores que nuestra librería obtiene con datos reales que dispone la Compañía.

- Memoria y reuniones: Redactado constante de la documentación. Se hará durante todo el transcurso del proyecto. Las reuniones con el director del proyecto también estarán incluidas aquí, siguiendo la metodología *SCRUM*.

- Final del proyecto: Ha concluido el día 13 de junio del 2011.

4.1.3. Diagramas de Gantt

Las figuras 5 y 6 representan los diagramas de *Gantt* para cada una de las planificaciones temporales.

4.1.3.1. Diagrama previsto

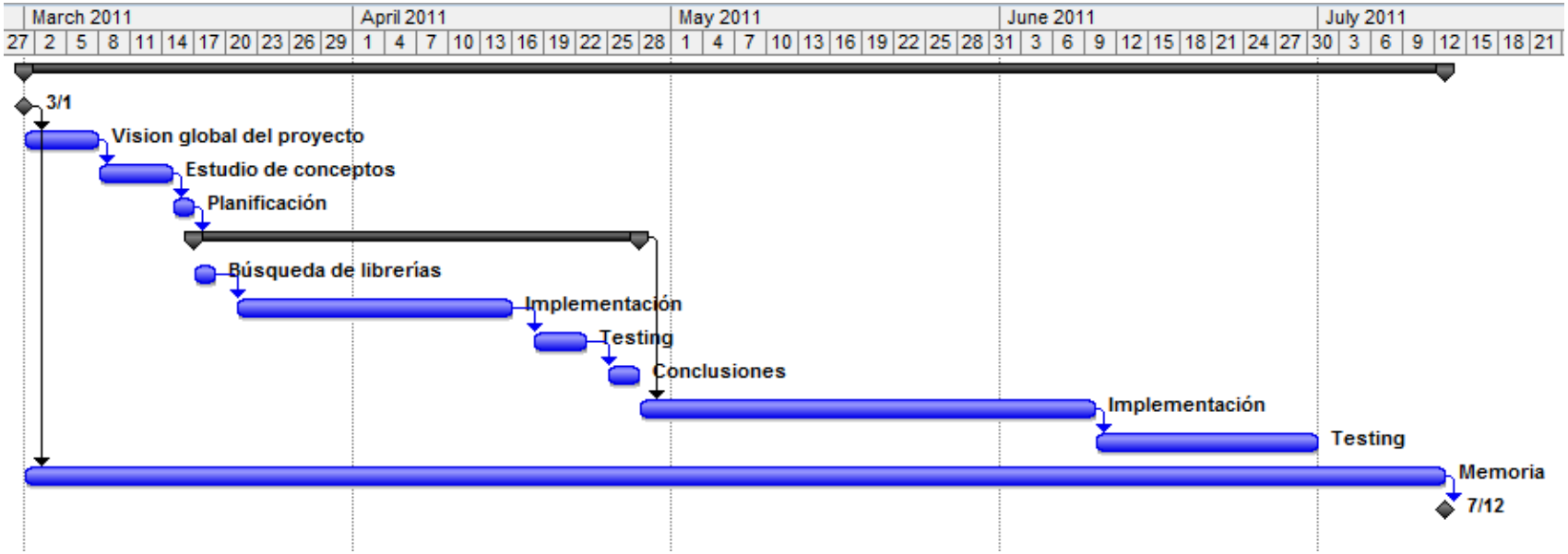

Figura 5

4.1.3.2. Diagrama final

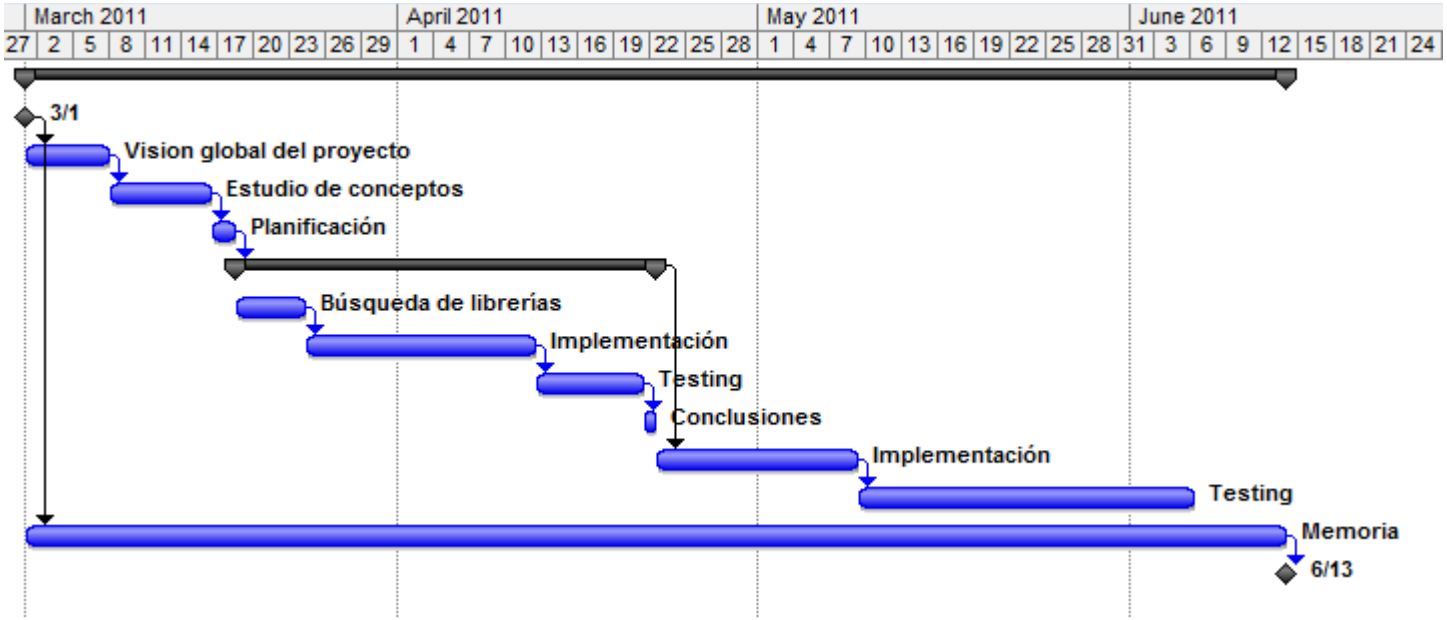

Figura 6

4.1.4. Desviaciones temporales

Como se puede apreciar en la tabla 2 ha habido una variación significativa respecto a la duración total del proyecto. Estos cambios son los que se reflejan a continuación:

- Se ha ampliado en 10 horas el tiempo destinado a estudiar los conceptos que se trabajarán en el proyecto, debido a que se ha considerado insuficiente el tiempo establecido inicialmente.

- Aumento de 11 horas en la búsqueda de librerías candidatas a ser utilizadas. Se pretendía hacer una búsqueda exhaustiva y se creyó conveniente aumentar el número de horas.

- Se ha reducido en 41 horas la implementación de los dos proyectos necesarios para hacer el *Benchmarking.* Se preveía una mayor complicación en el desarrollo de este procedimiento y se estiró el número de horas.

- Incremento de 12 horas en el proceso del *Testing*. Debido a que era básica la elección de la librería, se desarrollaron más gráficos de los que inicialmente estaban previstos.

- Reducción de 12 horas en las conclusiones sobre que librería se usará. El incremento en el *Testing* supuso un aumento en la cantidad de datos obtenidos facilitando así la elección de la librería.

- Decremento de 121 horas en la implementación final de la librería. Debido a la abundante documentación que dispone *DotNumerics* en la red, la codificación resultó ser más sencilla de lo esperado.

- Aumento de 30 horas en la verificación de cálculos de la librería. Aquí se concentraron el mayor número de problemas. Algunos datos no cuadraban con los que la Compañía disponía. En el apartado problemas encontrados [5] se podrá ver una explicación más detallada tanto de este como de los demás problemas que hayan aparecido durante el desarrollo.

- Reducción drástica de 126 horas en la elaboración final de la memoria. Se preveía un mayor número de horas destinadas a la memoria que al final no fueron necesarias.

[5] Ver página 91

4.2. Planificación económica

Para hacer la planificación económica se han tenido en cuenta tres aspectos:

- Salario otorgado al desarrollador.
- *Software* y *hardware* utilizado.
- Otros aspectos.

A su vez, la planificación económica puede ser real, situándose así en el contexto actual del desarrollador, o bien simulada, en el caso de que se tratase de un trabajador autónomo. Este estudio se ha hecho por dos razones:

- Comparar los costes totales del proyecto en ambos casos.
- Mejorar los conocimientos económico-empresariales adquiridos en la carrera.

4.2.1. Contexto actual

Sabiendo que el proyecto ha durado 450 horas y que se percibe 7€/h, la suma total asciende a 3.150€. Además debido a que se está en situación de convenio universidad-empresa, la Compañía tendrá que abonar un 14,7% del salario percibido a la universidad en concepto de gastos de gestión. Así pues los 3.150€ base se tendrán que sumar 463,05€ a la facultad con lo que la suma final asciende a 3.613,05€.

A nivel de *hardware y software* se dispone de un material de trabajo valorado en 1.500€ (se incluye el ordenador, monitores, teclado, ratón e impresora) con una amortización lineal a 4 años dando un coste anual de 375€. Dado que se ha necesitado 75 días para hacer el proyecto la amortización del equipo durante este periodo es de 77,05€.

Las licencias que se poseen son las siguientes:

Software	Precio
Windows 7 professional 64 bits	150€
Microsoft Visual Studio 2010 Ultimate	Comprar: 12.769€
	Renovación anual: 4.079€
Microsoft Office 2010	699€

Tabla 3

Con una amortización de 4 años para el Windows 7 y Microsoft Office 2010 y renovando anualmente el Visual Studio 2010 se obtiene un coste de 880,95€.

A todo eso se ha de sumar 450€, debido a que la librería que resultó ganadora del *Benchmarking* requiere de ese pago adicional para su uso comercial.

La suma de los inmuebles durante estos 75 días da un total de 45€ mientras que como material de oficina, la suma asciende a 60€. Los gastos de internet, luz y agua calculados son 150€.

Por tanto sumando todas las cantidades, el proyecto ha costado 5.276,05€.

4.2.2. Trabajador autónomo

En la siguiente tabla (ver tabla 4) se puede ver el coste anual de la empresa en 1.800 horas laborables:

Objeto de análisis	Coste
COSTE PERSONAL	
Salario (30€/h)	54.000€
Cotización S.S[6]	11.589,24€
ESPACIO TRABAJO	
Alquiler	7.200€
Teléfono/Internet	360€
Aseguradora	200€
Mobiliario	40€
HARDWARE	
Ordenador	175€
Periféricos	25€
Pantallas	100€
Impresora	20€
SOFTWARE	
Windows 7 Professional	50€
Visual Studio 2010	4079€

[6] 965,77€ mensuales incluido seguro médico.

Librería *DotNumerics*	450€
OTROS	
Desplazamientos	500€
Material extra	100€
COSTE ANUAL TOTAL	**78.888,24€**

Tabla 4

Las amortizaciones son las siguientes: *hardware* 4 años, *software* 4 años y mobiliario 8 años.

Así pues el coste de 450 horas que ha durado el proyecto es de 19.722,31€ para el autónomo frente a los 5.276,05€ en el estado actual del desarrollador.

5. Benchmarking

Sabiendo que los requisitos no funcionales eran la rapidez y un consumo mínimo tanto de memoria como de CPUs, se han realizado dos *tests* para saber qué librería es la que más conviene al proyecto. Los *tests* han sido los siguientes:

- Análisis aislado: Existe un único proceso que hace el cálculo del *Factor Risk*.
- Análisis en paralelo: Varios procesos ejecutan de forma paralela el cálculo anterior.

El estudio comparativo se ha realizado acotando el área a las siguientes librerías: *DotNumerics*[7]*, algLib*[8]*, dnAnalytics*[9] *y MatNet Numerics*[10]*.*

Condiciones del entorno de la prueba:

- *Hardware*:
  - Procesador: Intel® Core™2 Quad CPU Q8200 @ 2.33GHz 2.33 GHz
  - Memoria RAM: 4.00 GB.
- *Software*:
  - Sistema Operativo: Windows 7 Professional de 64 bits con Service Pack 1.
- Otras consideraciones:
  - Debido a la imposibilidad de aislar completamente nuestro proceso de la ejecución de otros en la CPU la tabla procesos del anexo[11] enseña los diferentes procesos que se encontraban en ejecución en el momento de la prueba.

El estudio se ha hecho en varias fases. Son las siguientes:

[7] http://www.dotnumerics.com/
[8] http://www.alglib.net/
[9] http://dnanalytics.codeplex.com/
[10] http://mathnetnumerics.codeplex.com/releases/view/56448
[11] Ver figura 20 página 95

Mostrar el tiempo de respuesta en segundos del cálculo del *Factor Risk* para cada una de las librerías, con unos datos extremos e intermedios para ver cómo se comportan cada una de éstas.

Esta operación implica de los siguientes pasos:

- Una inicialización de una matriz con varios registros.
- Una inicialización de un vector con los mismos registros.
- Una transposición del vector anterior.
- Dos multiplicaciones, una del primer vector y la matriz y el resultado de está por el vector transpuesto.

La media de registros está entre los 1.000 y los 3.000, por tanto se tratarán estos dos valores así como cuatro casos extremos que estarán situados en los límites inferiores y superiores.

En la segunda parte del análisis se obtendrá el consumo de cada una de las CPUs y de la memoria RAM utilizada. Los datos se expondrán de forma agrupada para facilitar la comparativa, al igual que con los tiempos de ejecución, en unos gráficos generados por Microsoft Excel 2007.

Finalmente, la tercera parte consistirá en analizar todos los datos y mediante descarte, escoger la librería a utilizar en el proyecto. La elección final se hará basándose los siguientes pesos:

- Tiempo de respuesta: 50%,
- Uso de CPU: 25%
- Consumo de RAM: 15%.
- Criterios subjetivos: 10% repartidos de la siguiente forma.
    - Documentación: Cuánta información y ejemplos se dispone en la página web oficial de la librería (6%).
    - Facilidad de uso: Cómo de intuitiva será su utilización (4%).

La elección de pesos no ha sido casual. Como se ejecutará en un entorno gráfico y se prevé que haya varios usuarios ejecutando el mismo proceso, es necesario que el tiempo de respuesta sea el que más valor tenga en la decisión final.

EL consumo de la CPU está en segundo lugar puesto que mejorar la máquina es más costoso que aumentar la RAM.

A la hora de mostrar los datos, el proceso será el siguiente: primero se darán los resultados corriendo el proceso de forma aislada, y una vez finalizado el primer estudio, se pasará a la ejecución paralela mediante 10 *threads* en una misma máquina.

5.1. Análisis aislado

5.1.1. Tiempo de respuesta

La manera de calcular el tiempo en un proyecto en C# es mediante la clase System.Diagnostics. Stopwath. Cuando se inicia el proyecto, el método Start () activa este medidor y mediante el método Stop () se detiene una vez se hayan completado todos los cálculos. Esto es:

```
Stopwatch time = new Stopwatch();
time.Start();

//CÓDIGO

time.Stop();
Console.WriteLine("Total Time: {0}", time.Elapsed);
```

En la tabla 5 se muestran los tiempos de ejecución en segundos de cada una de las cuatro librerías y un gráfico (ver gráfico 1) con los datos recogidos de la misma:

Registros/Biblioteca	50	500	1.000	3.000	8.000	15.000
DotNumerics	0.0215102	0.0332984	0.0639101	0.3856661	2.5883645	29.5690919
dnAnalytics	0.0196734	0.0343442	0.0768381	0.0768381	5.3455686	33.3549656
MathNet	0.0328947	0.0483441	0.1085874	0.908924	8.1329037	47.577799
algLib	0.0687342	8.9292041				

Tabla 5

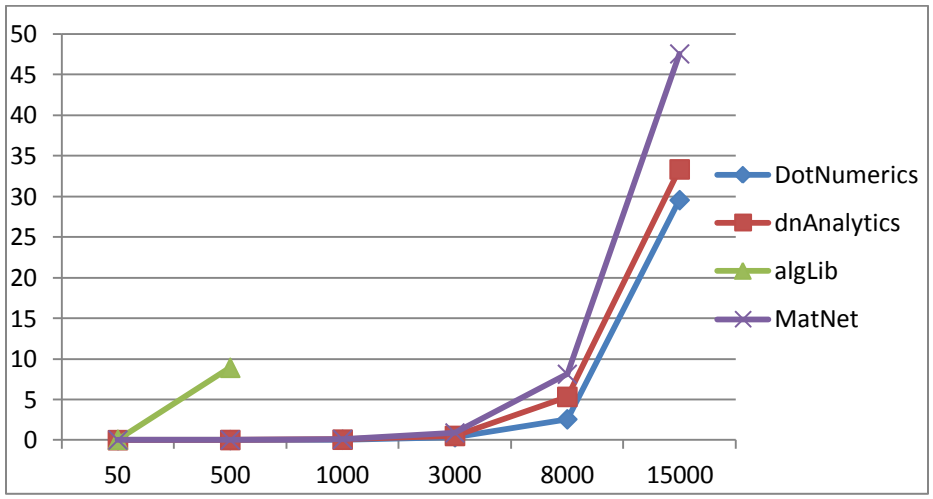

Gráfico 1

Tanto *DotNumerics* como *dnAnalytics* se comportan de manera similar, no obstante parece ser que *DotNumerics* se comporta mejor a medida que se añaden más registros ya que en el caso extremo de 15.000 registros, la diferencia es de prácticamente 4 segundos. La librería *algLib*, con tan solo la inicialización de 500 registros de la matriz se obtiene 9 segundos, unos datos que hace que se excluya del estudio. Finalmente *MatNet* muestra un rendimiento similar a las dos primeras en registros bajos pero a partir de los 8.000, la curvatura crece mucho más que las otras dos posibles candidatas. Esta última librería sería una candidata a pasar al siguiente estudio si no fuera por la tabla de tiempos que se obtiene en el análisis *Multi-Thread*. Posteriormente[12] se explicará las razones de su exclusión.

5.1.2. Uso de la CPU

Una vez finalizada la primera fase del análisis y contando únicamente con las librerías *DnAnalytics* y *DotNumerics* se pasará a obtener el uso de la CPU que hacen estas dos librerías.

Para realizar estos cálculos, al igual que con el uso de la memoria, se ha realizado un proyecto Java en Netbeans y se ha incluido la librería sigar.jar[13] a éste.

La razón de escoger esta librería es que en ella está todo lo necesario para saber los datos que nos producen cada una de las librerías a nivel de la CPU.

[12] Tabla 7 página 50
[13] http://support.hyperic.com/display/SIGAR/Home

El programa correrá de forma paralela a la ejecución del principal e irá volcando los datos que vaya obteniendo a un archivo de Microsoft Excel para poder comparar los datos. Un ejemplo del *output* que produce el programa se puede ver en la tabla 11 que se encuentra en el anexo[14].

La máquina en la que se está corriendo tiene 4 CPUs. Se muestra el % de uso de cada una de éstas, así como el consumo total. El objetivo de este y del siguiente apartado no es mostrar todas las tablas, sino enseñar los datos en gráficos para hacerlo de forma más visual.

En cada una de las tablas siempre se muestran dos picos en el uso de la CPU. El primero de ellos se produce cuando se inicia el programa, el otro cuando se da por finalizado el programa principal y la máquina libera la memoria ocupada por el proceso.

La última fila de la tabla muestra el valor máximo del uso de la CPU así como la media de todos los registros. Esta última fila será la que se usará para realizar los gráficos de rendimiento.

Recordar que tanto este análisis como en el siguiente, se ha hecho usando únicamente dos librerías, la *DotNumerics* y la *dnAnalytics*. La *algLib* y la *MathNet* han sido descartadas por el tiempo de respuesta para inicializar una matriz.

A continuación se muestran todos los gráficos para cada una de las diferentes cantidades de registros de la matriz. El resultado final de ejecutar 10 veces cada uno de los procesos y realizar la media de éstos será el que figure.

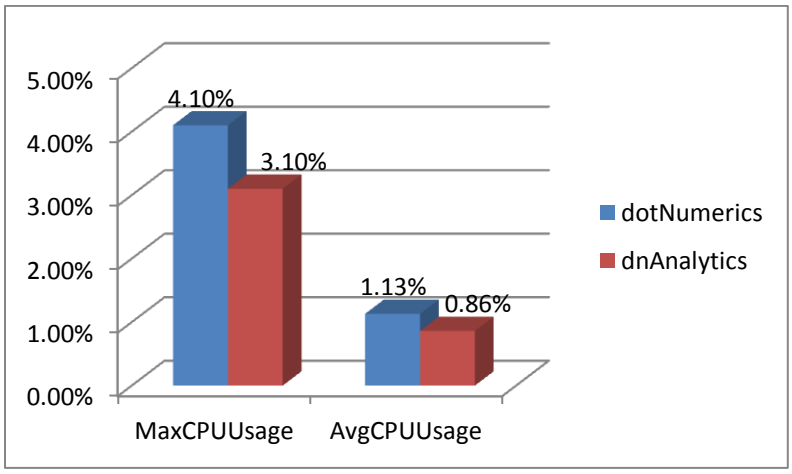

Gráfico 2

[14] Ver página 96

En el caso inicial de 50 registros se observa como *dnAnalytics* se comporta mejor que *DotNumerics*, no obstante esta carga no es para nada representativa (ver gráfico 2)

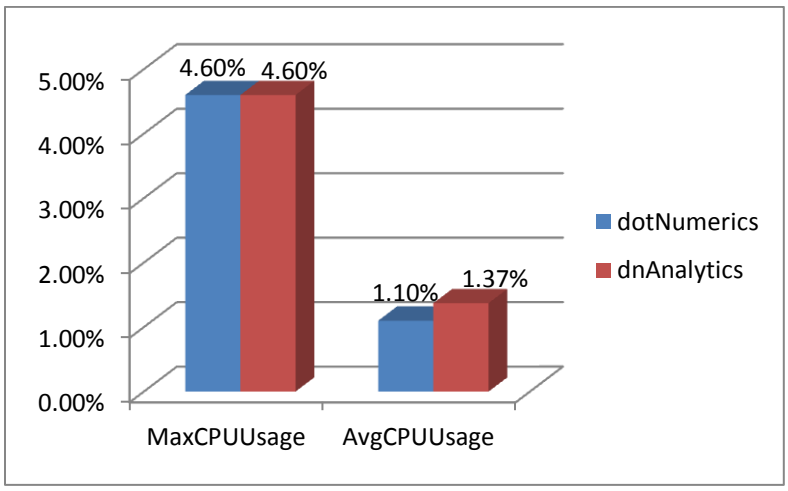

Gráfico 3

Se muestran unos datos muy parejos, el pico es exactamente igual en ambos, no obstante hay una pequeña variación con la media (ver gráfico 3)

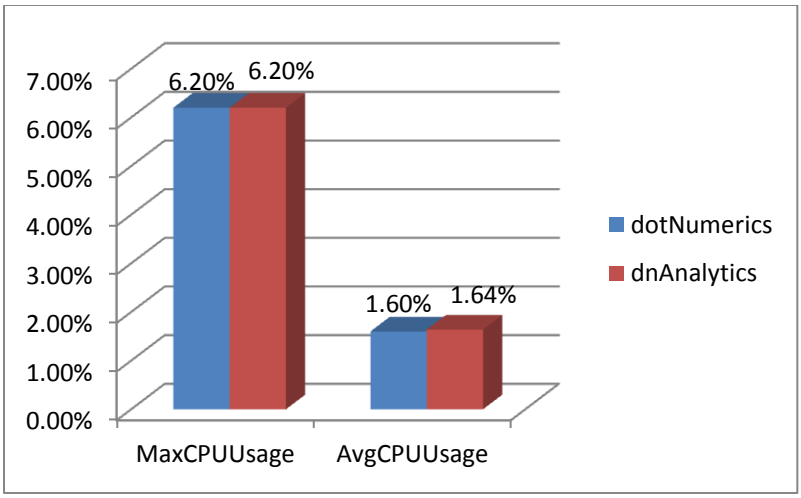

Gráfico 4

Resultados parejos en el caso de 1.000 registros con una ligera diferencia a favor de *DotNumerics* de tan solo 0,04% (ver gráfica 4).

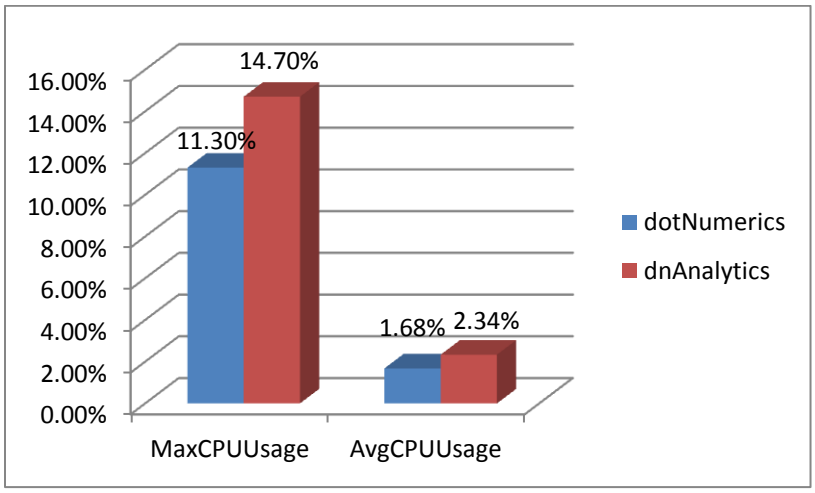

Gráfico 5

La gráfica 5 es la primera a considerar, tanto por el número de registros como por la tendencia que marcan los valores. El uso máximo de CPU para *DotNumerics* es 11,30% mientras que para *dnAnalytics* es 14,70%, además la media para la primera librería es de 1,68% mientras que para la segunda 2,34%.

Con estos datos, la tendencia al alza que tiene *dnAnalytics* respecto a *DotNumerics* a medida que se van añadiendo más registros es clara.

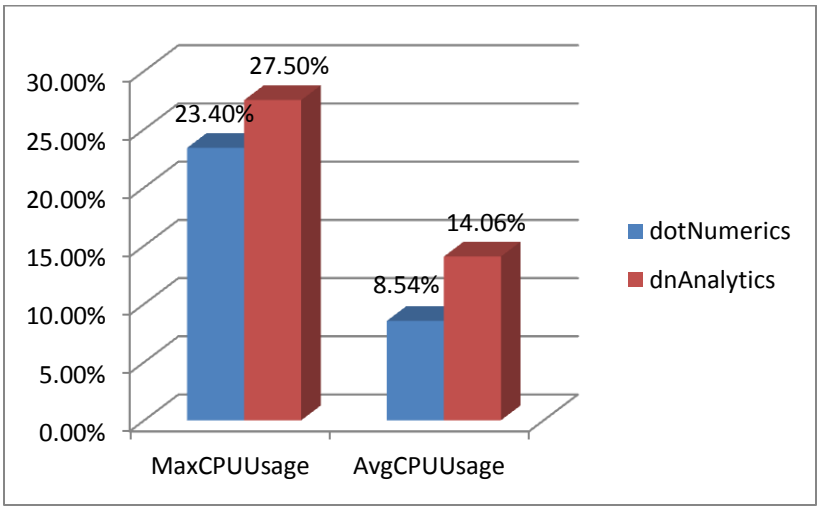

Gráfico 6

A medida que se aproxima el caso límite de 15.000 registros, las diferencias se van acentuando entre ambas. *DotNumerics* con 23,40% de pico y 8,54% de media, es bastante inferior a *dnAnalytics* con 27,40% y 14,06% respectivamente (ver gráfico 6).

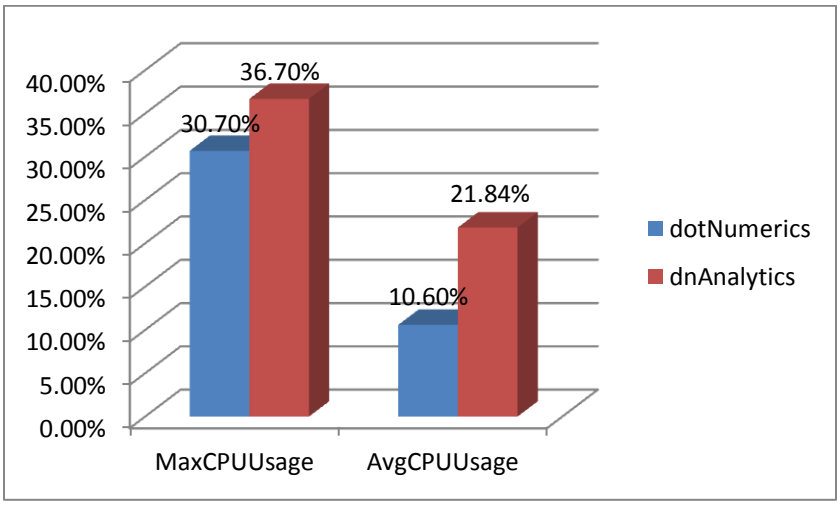

Gráfico 7

Finalmente, en el caso extremo de 15.000 registros (ver gráfico 7) *DotNumerics* consume mucha menos CPU que *dnAnalytics* para realizar las mismas operaciones. Los valores de DotNumerics con 30,70% y 10,60% son claramente inferiores a los 36,70% y 21,84% de la última.

Una vez finalizados los primeros test se puede afirmar que *DotNumerics* se comporta mejor que *dnAnalytics* a medida que se va incrementando el número de registros de la matriz.

5.1.3. Uso de la memoria

Un ejemplo del *output* de este test se muestra en la tabla 12 del anexo[15].

El primer gráfico será el cálculo de las RAM deltas (diferencia entre un intervalo y el siguiente) y el segundo corresponderá al uso de la RAM en un instante de tiempo concreto.

Será obligatorio redefinir los valores para que ambas gráficas empiecen con el mismo consumo de RAM porque es posible que por la ejecución de otros procesos, haya una cantidad de RAM usada que haga descuadrar el gráfico. Se puede generalizar que *no importa el valor absoluto de la RAM si no el valor que usa únicamente la librería*.

[15] Ver página 97

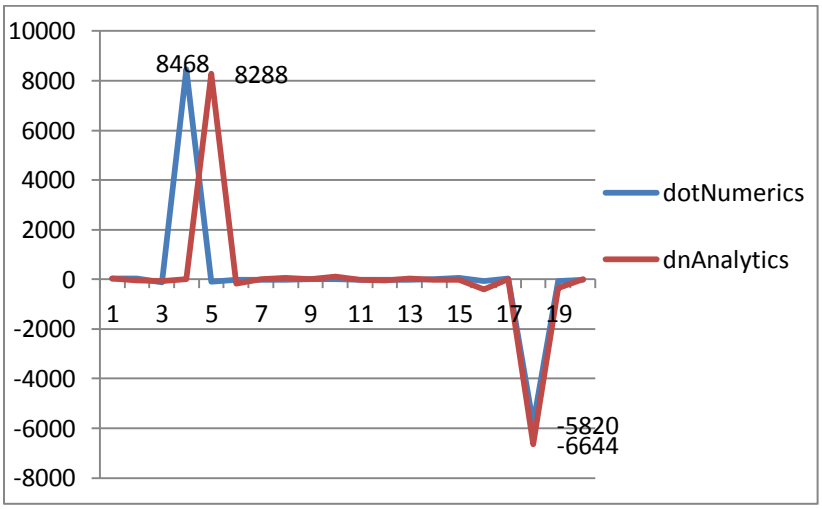

Gráfico 8

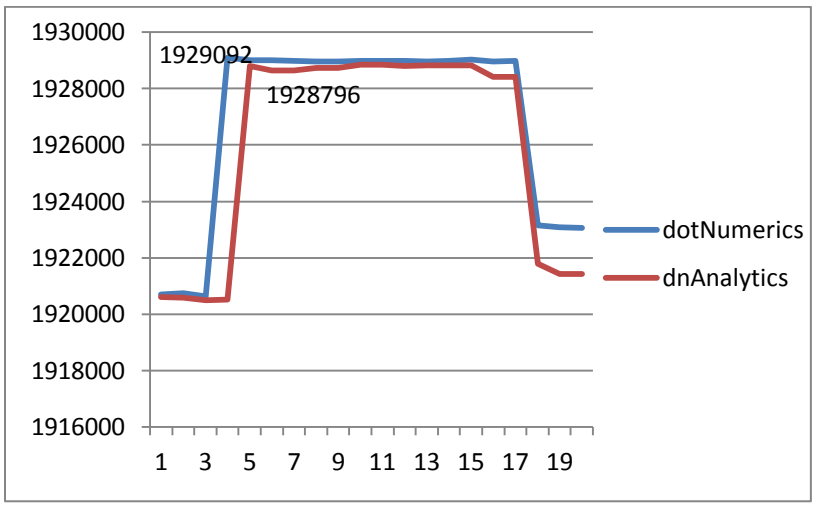

Gráfico 9

50 registros no es una medida a tener en cuenta (ver gráficos 8-9), no obstante se puede apreciar que ambas librerías usan prácticamente la misma memoria RAM pero existe una pequeña diferencia que favorece a *dnAnalytics*.

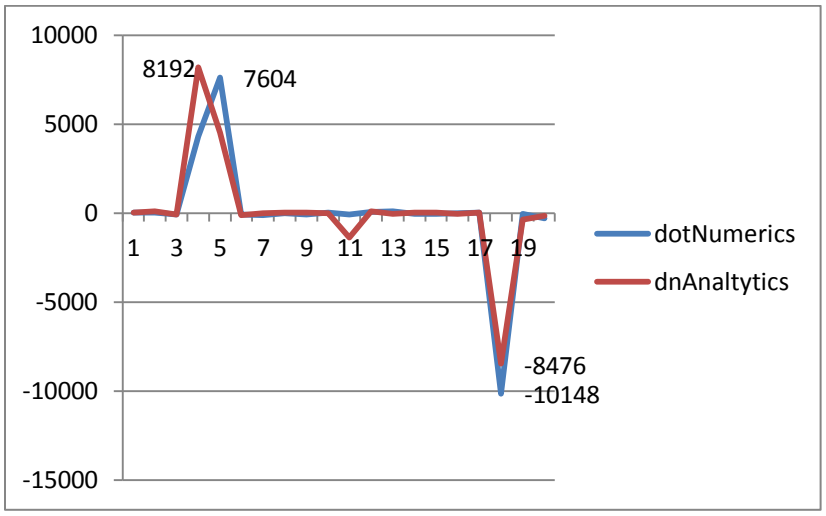

Gráfico 10

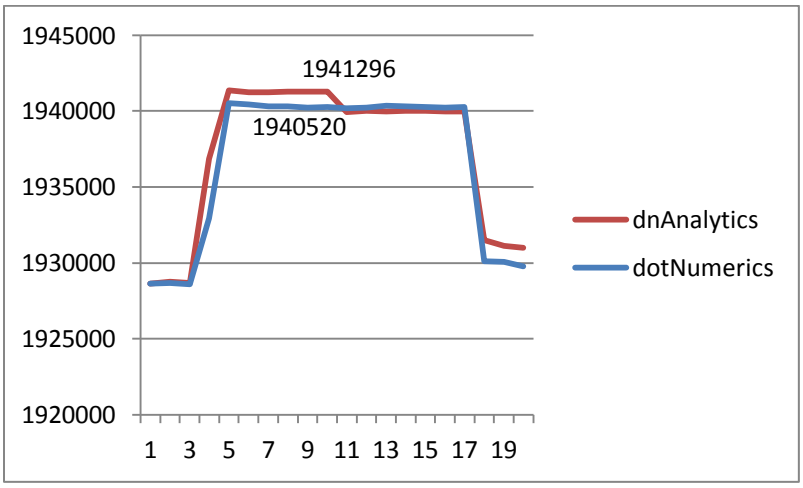

Gráfico 11

Con 500 registros (ver gráficos 10-11) se ve un mayor uso de RAM en *DotNumerics*. Como se ve en la gráfica 11, *DotNumerics* usa más a lo largo de todo el proceso.

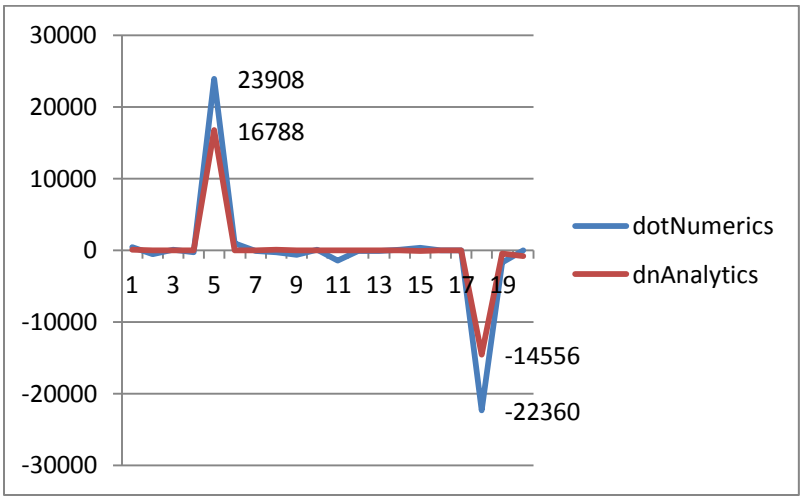

Gráfico 12

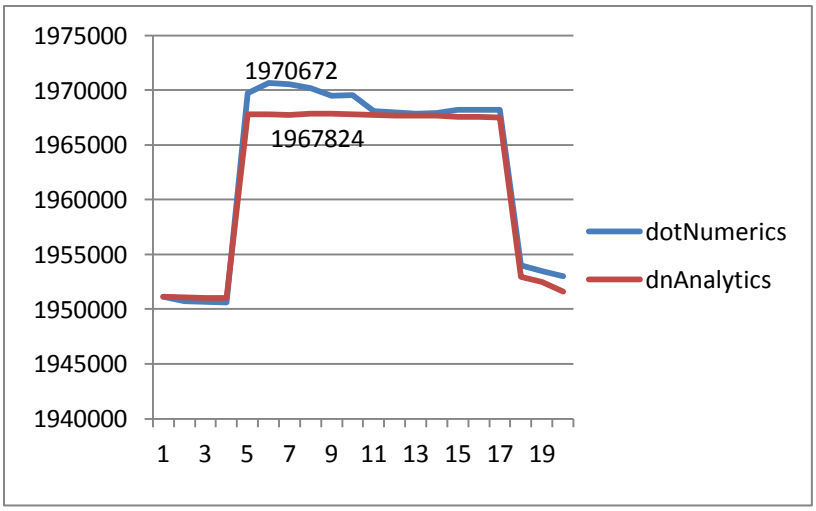

Gráfico 13

En los gráficos 12 y 13 se comprueba un mayor consumo de RAM en *DotNumerics*. Se ha de tener en cuenta que el incremento de *DotNumerics* es más pronunciado que la de *dnAnalytics*, a parte esta última parece más estable en cuanto a los recursos necesarios.

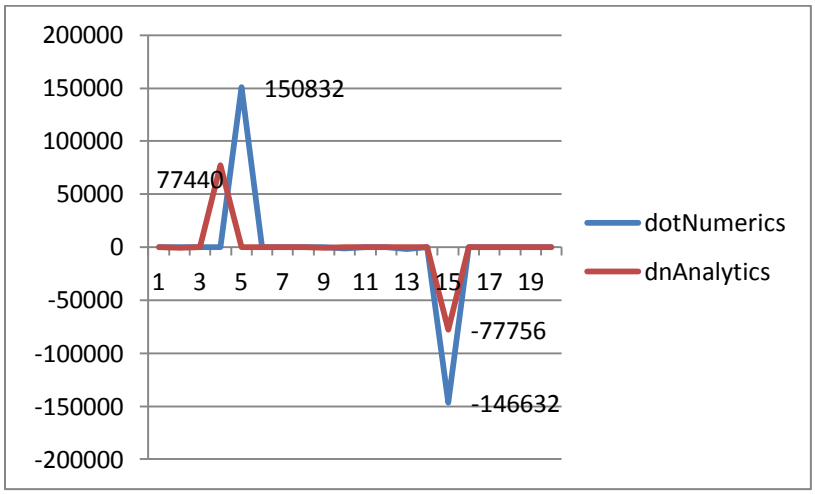

Gráfico 14

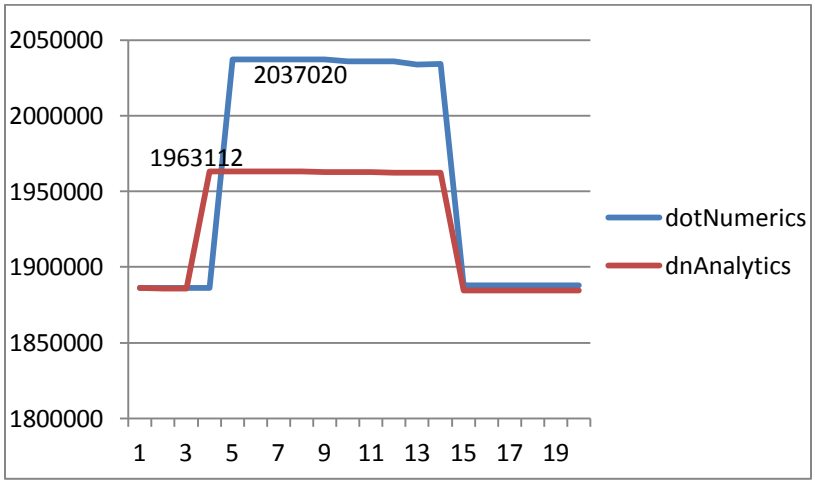

Gráfico 15

La gráfica 14 es la primera que se tiene que tomar en consideración. En la gráfica 15, cuadrando *dnAnalytics* al mismo nivel que *DotNumerics* para simular el mismo inicio de RAM, se ve claramente como *dnAnalytics* consume menos memoria que *DotNumerics*.

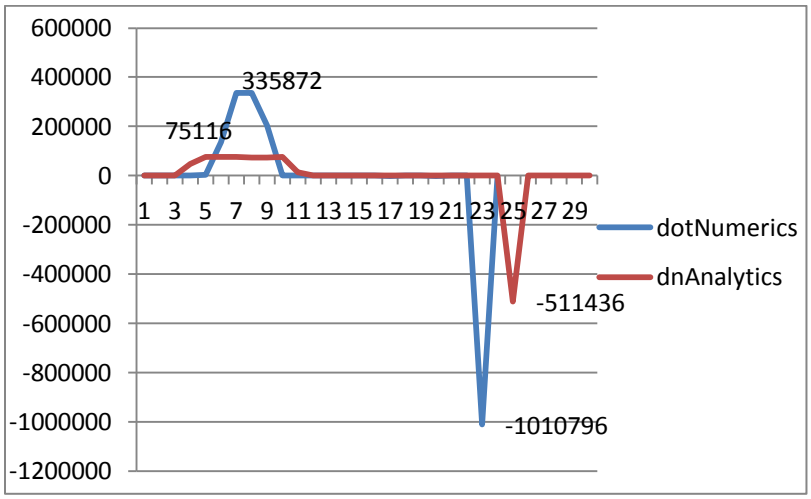

Gráfico 16

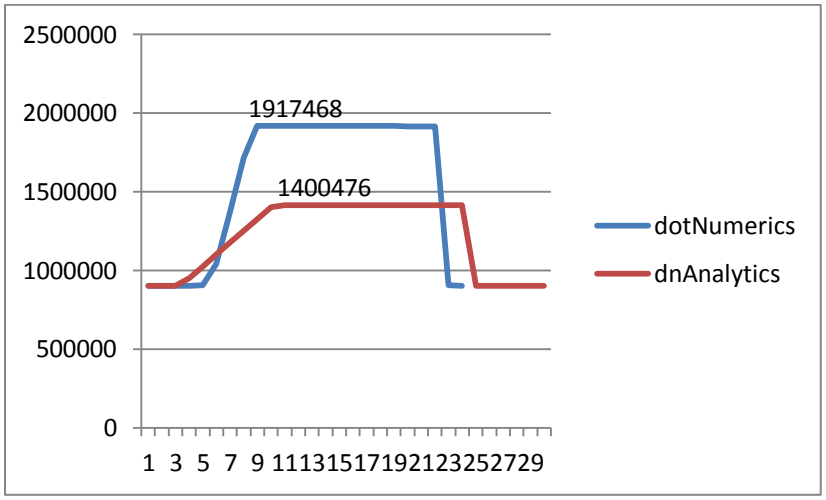

Gráfico 17

La punta alcanzada por *DotNumerics* es muy superior a *dnAnalytics*. Además en la gráfica 17 se ve claramente como la curva de crecimiento de *DotNumerics* es más pronunciada que la de *dnAnalytics*.

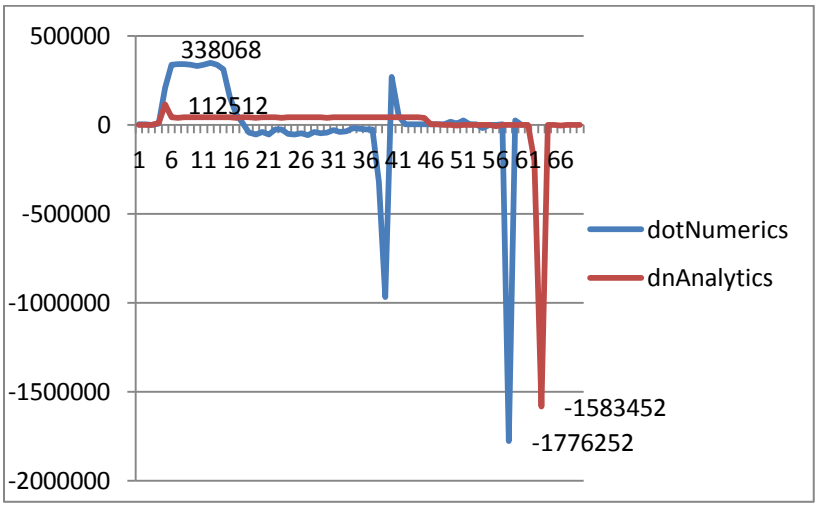

Gráfico 18

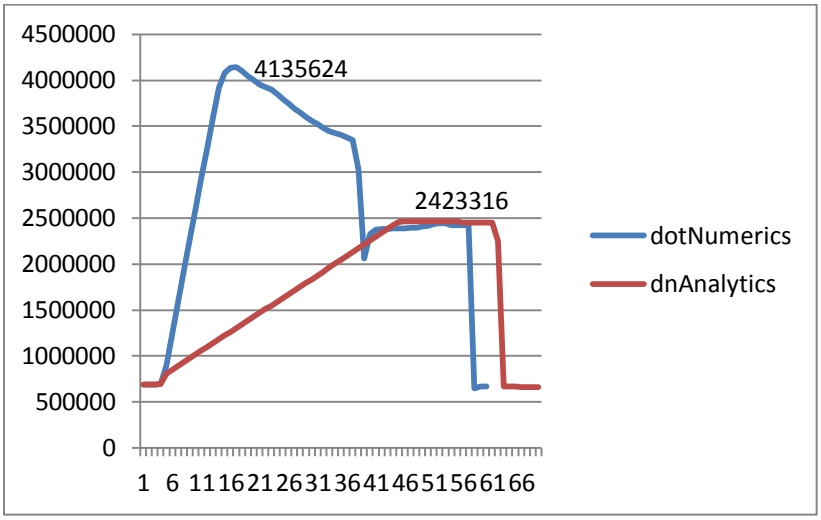

Gráfico 19

En la gráfica 19 se muestra cómo se comportan las dos librerías. Un crecimiento más rápido de memoria RAM usada para la *DotNumerics* pero por el contrario, *dnAnalytics* crece más lentamente, produciendo menos uso de memoria.

Se puede concluir que *DotNumerics* a medida que van creciendo los registros, aumenta su velocidad, pero como dato negativo, esto repercute en un mayor uso de memoria RAM.

A continuación se muestran los datos de forma conjunta para dar por concluido el primer análisis (ver tabla 6 y gráficos 20,21)

5.1.4. Visión global

Librería	50		500		1.000		3.000		8.000		15.000	
	CPU	RAM	CPU	RAM	CPU	RAM	CPU	RAM	CPU	RAM	CPU	RAM
dnAnalytics	0.86%	5.402	1.37%	8.561	1.64%	11.528	2.34%	41.782	14.08%	303.355	21.84%	991.435
DotNumerics	1.13%	6.157	1.10%	7.987	1.60%	11.887	1.68%	75.636	8.54%	651.765	10.68%	2.035.045

Tabla 6

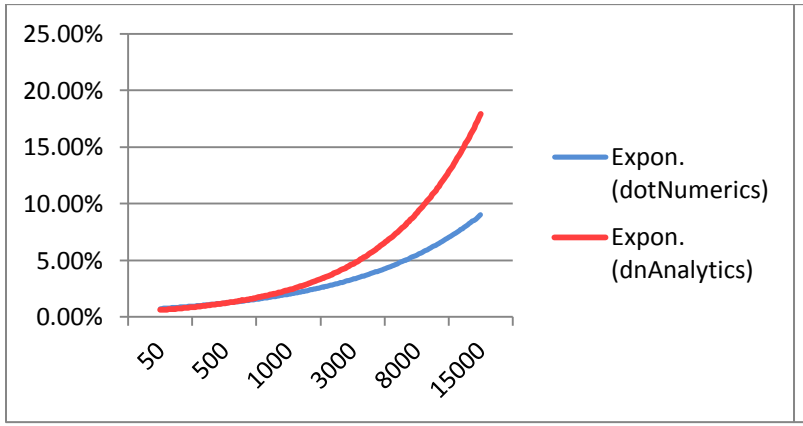

Gráfico 20

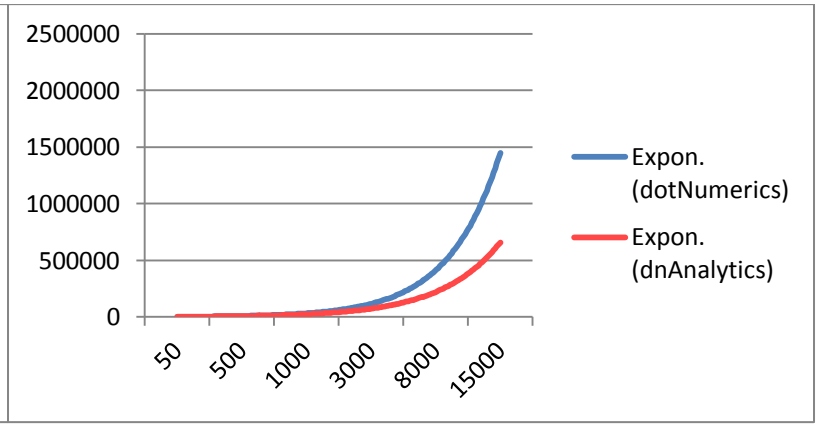

Gráfico 21

Mientras que *DotNumerics* es más rápida haciendo que el consumo de la CPU sea menor, *dnAnalytics* sale ganando en consumo de RAM total sacrificando tiempo de cálculo. Posteriormente, basándonos en los pesos, se decidirá cuál sería nuestra librería para el proyecto para un único *Thread.*

5.2. Análisis multithread

Se ejecutaran 10 *threads* del mismo proceso para ver cómo se comportan en una situación similar a la que se encontrará habitualmente. Los registros de las matrices variaran ya que el cálculo de 10 *threads* con 8.000 y 15.000 registros ha sido imposible realizarlos debido a unas limitaciones en el *hardware*. Los primeros test (50-500-1000-3000) se mantendrán y se añadirán de nuevos, un *test* con 5.000 registros y otro para simular una situación lo más cercana posible donde habrá variedad de registros.

Debido a que la mayoría de los cálculos se realizarán como media entre los 1.500 registros y los 3.000, la elección de los registros es la siguiente:

- 1 registro de 50
- 1 registro de 500
- 3 registros de 1.500
- 3 registros de 3.000
- 1 registro de 8.000
- 1 registro de 15.000

Por tanto se puede apreciar que se tienen 10 registros, 6 de éstos se concentran en los casos intermedios y los 4 restantes representan extremos por ambos lados, tanto en los casos menos costosos como en los que más.

5.2.1. Tiempo de respuesta

Como en el anterior análisis, el tiempo de respuesta será la principal fuente de información y la que tendrá un mayor peso en la elección final. La tabla 7 muestra estos tiempos:

Registros/Biblioteca	50	500	1.000	3.000	5.000	VARIOS
DotNumerics	0.0445896	0.1267235	0.2997478	1.9555369	6.0926145	25.194796
dnAnalytics	0.0552426	0.1650445	0.4002357	3.1078789	8.8817889	35.898223
MathNet	0.1047144	0.2136917	0.5079932	6.5447783	39.524877	*Sin memoria*
algLib	0.0789898	20.544775				

Tabla 7

Se aprecia como de nuevo *algLib* resulta muy desfavorecida respecto a las otras dos librerías por tanto, queda descartada del análisis. En el apartado de un único *Thread* se dijo que además de ésta última, *MathNet* también quedaba descartada.

Si bien *MathNet* está muy alejada de las otras dos librerías ya para 5.000 registros, era otra opción para valorar y por tanto una posible candidata. No obstante en el último análisis fue imposible realizar el cálculo debido a falta de memoria.

Para verlo de forma más clara, a continuación se puede ver en el gráfico 22 con los datos obtenidos en segundos de los tiempos de ejecución.

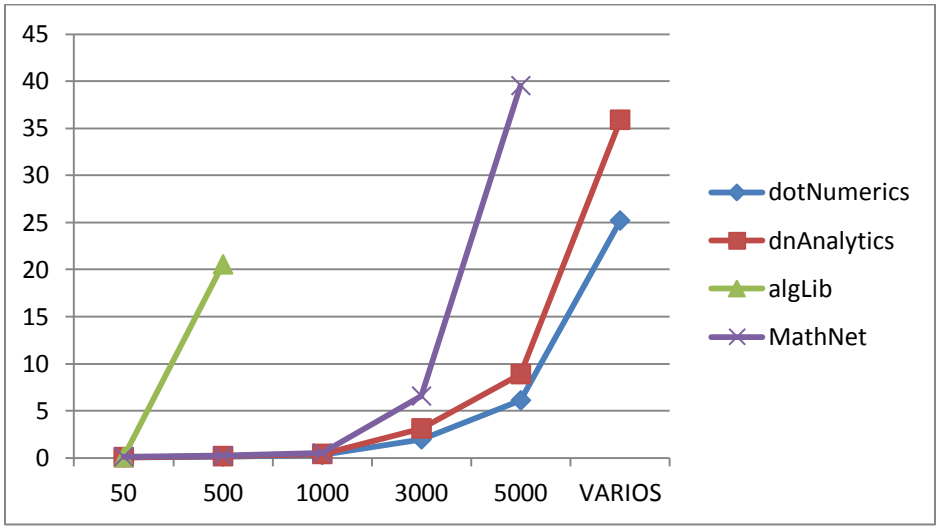

Gráfico 22

5.2.2. Uso de la CPU

A continuación se incluirán los gráficos de uso de la CPU para la ejecución de múltiples *threads*. El proceso será el mismo, se realizará diez veces cada una de las medidas para obtener un resultado lo más próximo posible a la realidad.

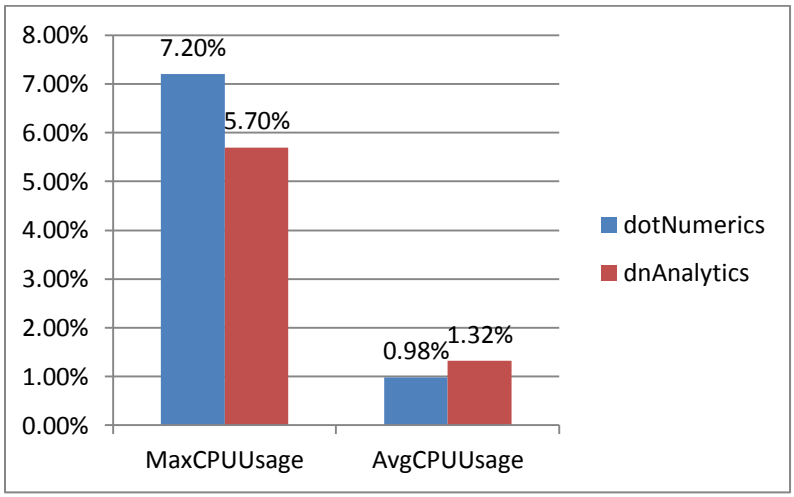

Gráfico 23

Como en el caso anterior sale perjudicada *DotNumerics* no obstante, parece ser que la media de uso le favorece.

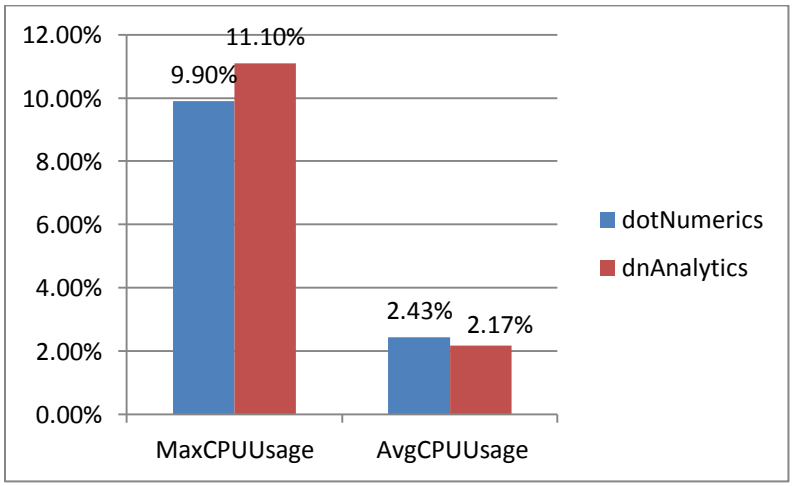

Gráfico 24

Si bien para un único *Thread* la diferencia empezaba a ser notoria con 3.000 registros, aquí con tan sólo 500, el máximo de *DotNumerics* es de 9.90% mientras que el de *dnAnalytics* es de 11.10%. La media no ha variado y siguen siendo muy parejas ambas librerías (ver gráfico 24).

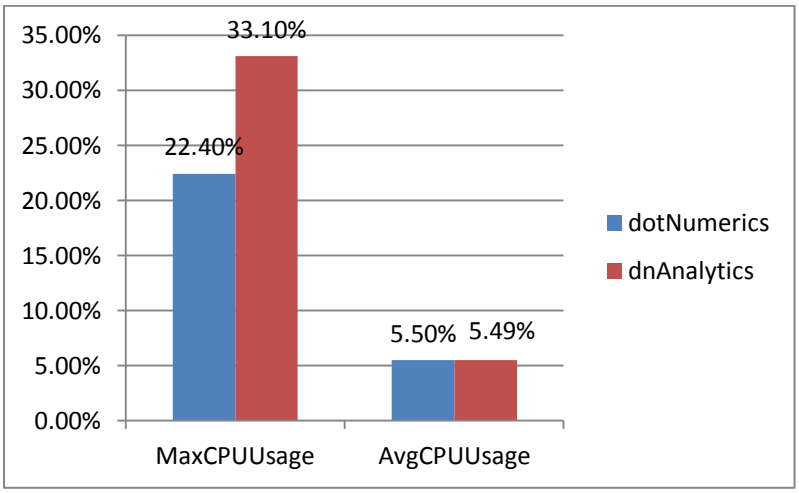

Gráfico 25

En el gráfico 25 el consumo es mucho más elevado en *dnAnalytics* como caso máximo, mientras que en la media, ambas librerías se mantienen igualadas.

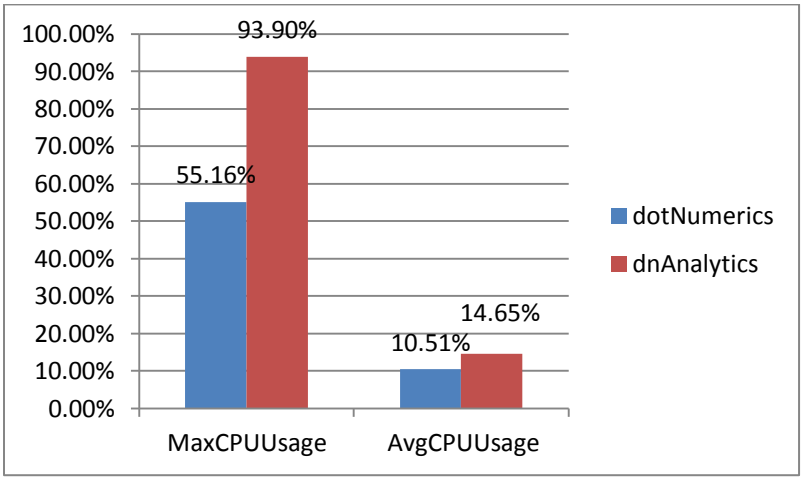

Gráfico 26

Se observan grandes diferencias en el uso de la CPU. Si bien la media continua siendo relativamente pareja, el consumo máximo es muy superior el de *dnAnalytics* respecto al de *DotNumerics* (ver gráfico 26)

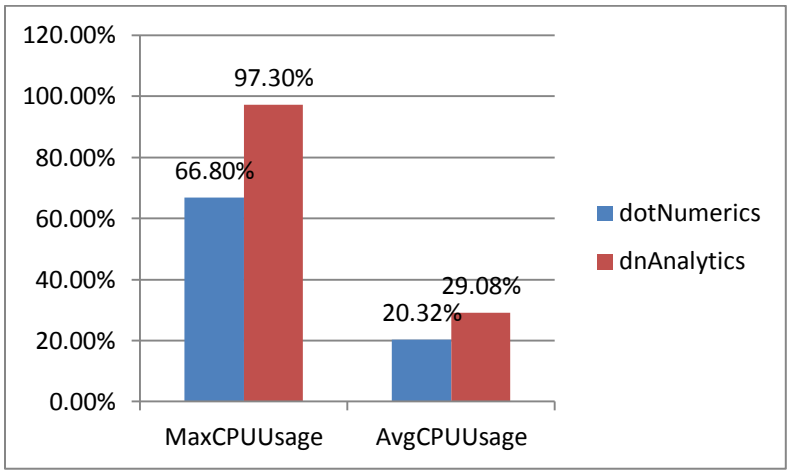

Gráfico 27

DotNumerics presenta un gran incremento, donde se pasa de un máximo de 55.16% a otro de 66.80%. El consumo medio, por el contrario ha aumentado mucho más para *dnAnalytics*, un 15%, a diferencia de *DotNumerics* donde el crecimiento es de un 10% (ver gráfico 27)

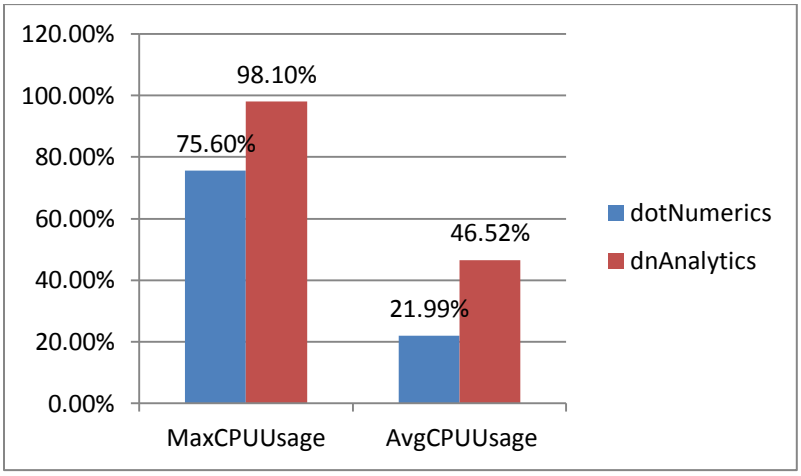

Gráfico 28

Como bien se ha dicho anteriormente, en una situación real, no se prevé una homogeneidad en los registros, por lo tanto, este último test es el que más se acerca a una posible realidad. Se aprecia como en el último caso, *DotNumerics* resulta ganadora con mucha diferencia respecto *dnAnalytics* (ver gráfico 28)

5.2.3. Uso de la memoria

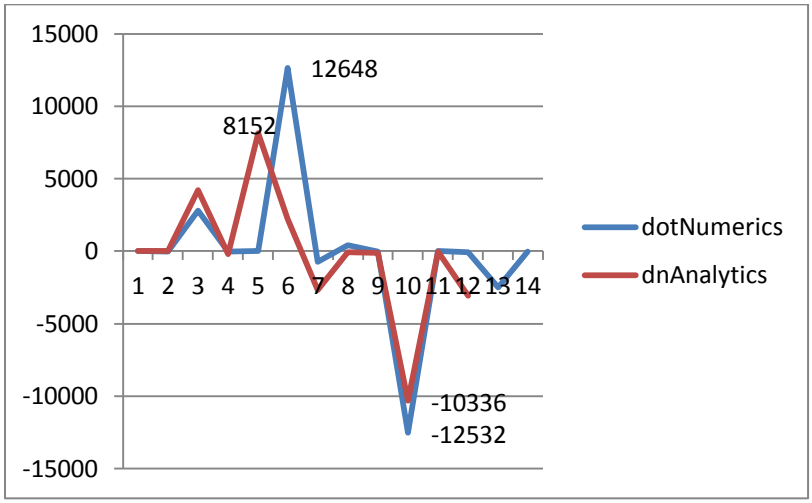

Gráfico 29

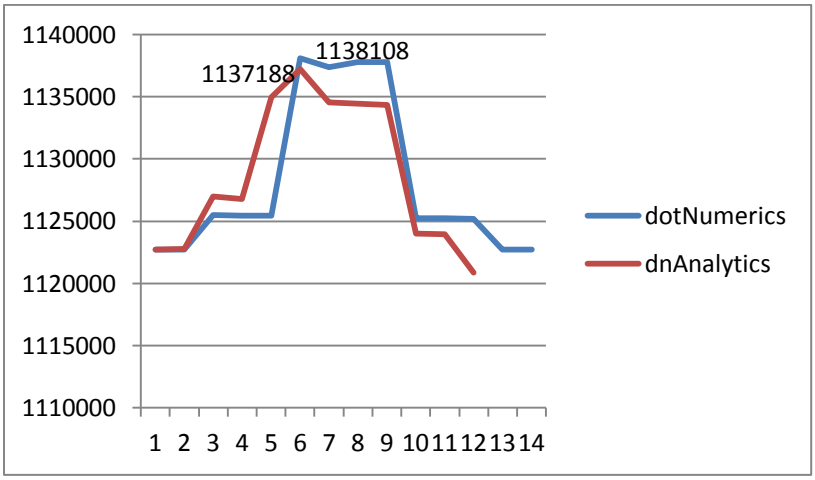

Gráfico 30

Como en el caso de un único *thread*, *dnAnalytics* se comporta mejor con menos registros y en el caso de ser *multi-thread* se comporta aún mejor que con uno de solo (ver gráficos 29-20)

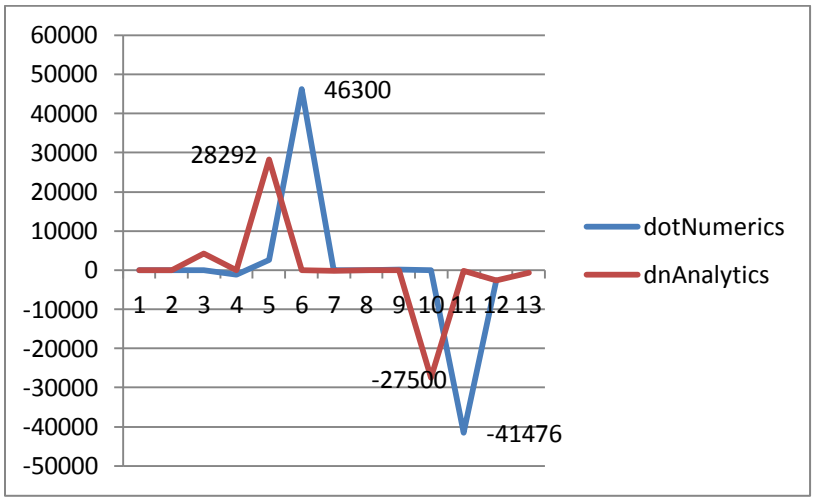

Gráfico 31

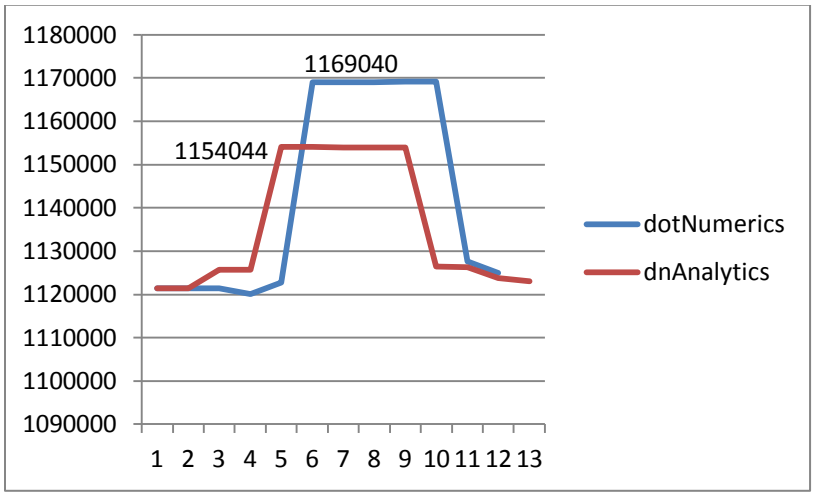

Gráfico 32

Se mantiene la tendencia inicial de un mejor comportamiento de *dnAnalytics* con pocos registros (ver gráficos 31-32)

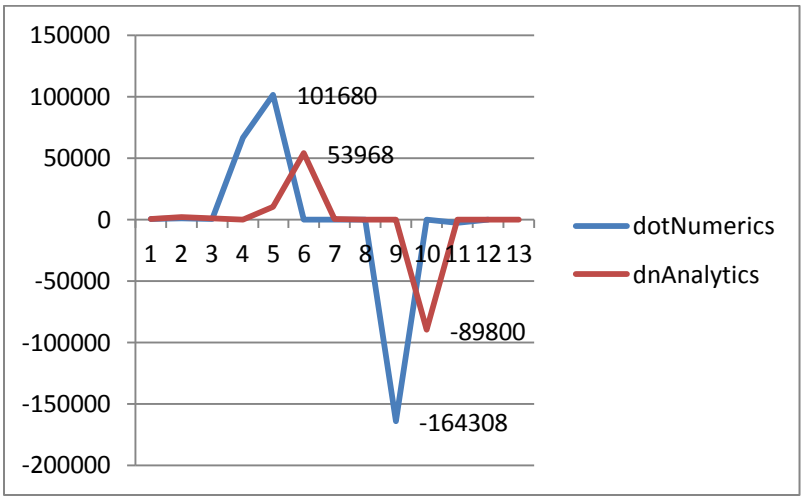

Gráfico 33

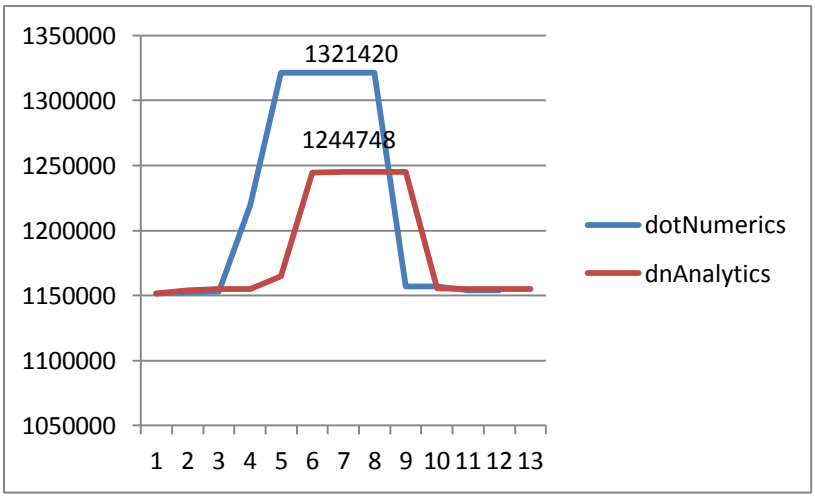

Gráfico 34

A medida que van creciendo los registros, se observa como *dnAnalytics* se comporta mejor que *DotNumerics*, tanto en las deltas como en el uso de la RAM (ver gráficos 33-34)

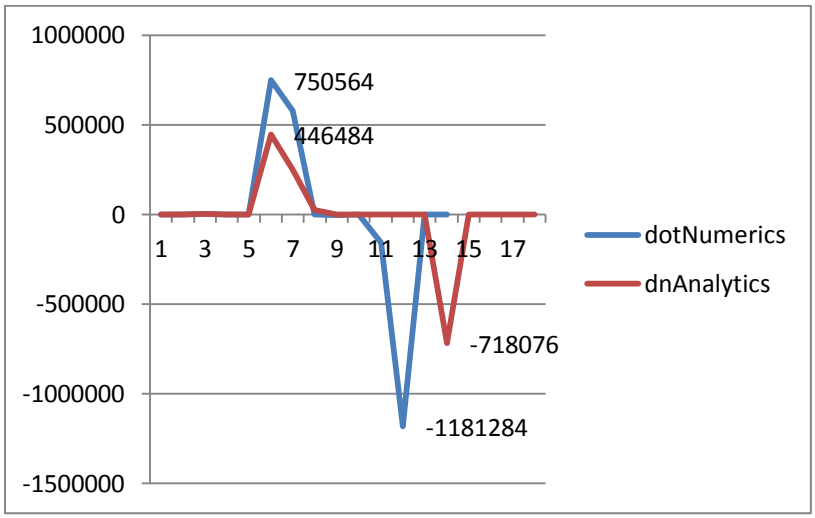

Gráfico 35

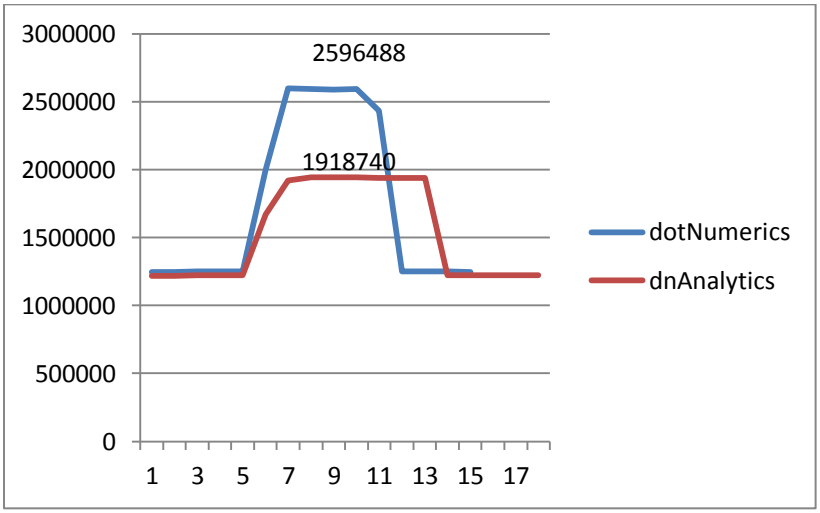

Gráfico 36

Los resultados son idénticos al primer punto pero con datos mayores, por tanto se deduce que con una cantidad de registros de 5.000, consumirá mayor RAM *DotNumerics* que *dnAnalytics*.

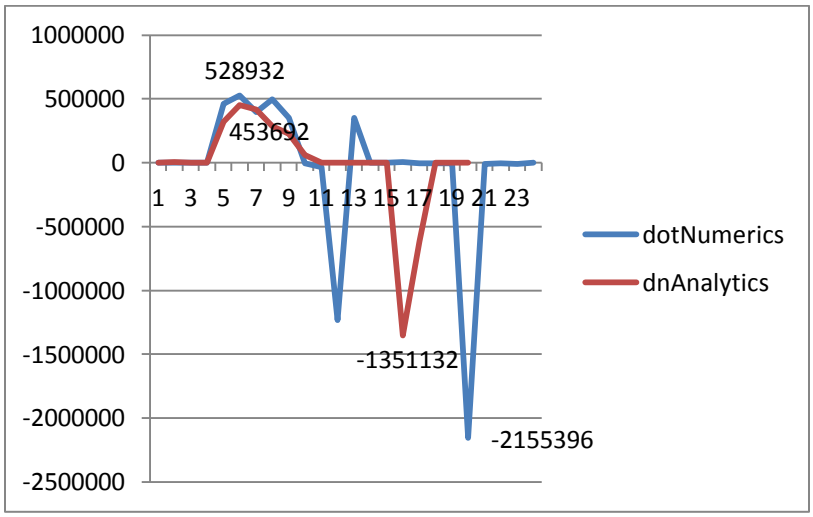

Gráfico 37

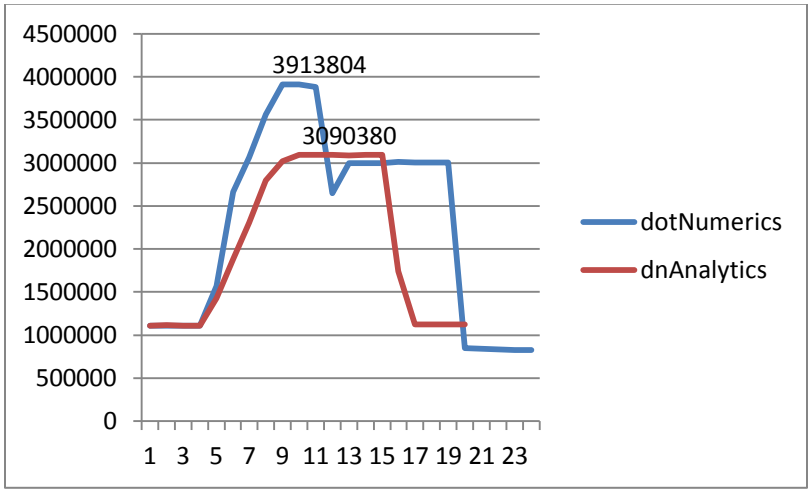

Gráfico 38

En este caso ocurren dos cosas interesantes. La primera de ellas es que se mantiene la tendencia tal y como se decía en el apartado anterior. La segunda es que el comportamiento diferente entre las dos librerías que anteriormente pasaba con 15.000 registros, aquí se reproduce con los 5.000 registros. Mientras que *DotNumerics* realiza una subida muy alta de RAM al principio para luego liberar y volver a usar más, *dnAnalytics* lo hace de forma diferente ya que únicamente libera una vez finalizado todo el proceso.

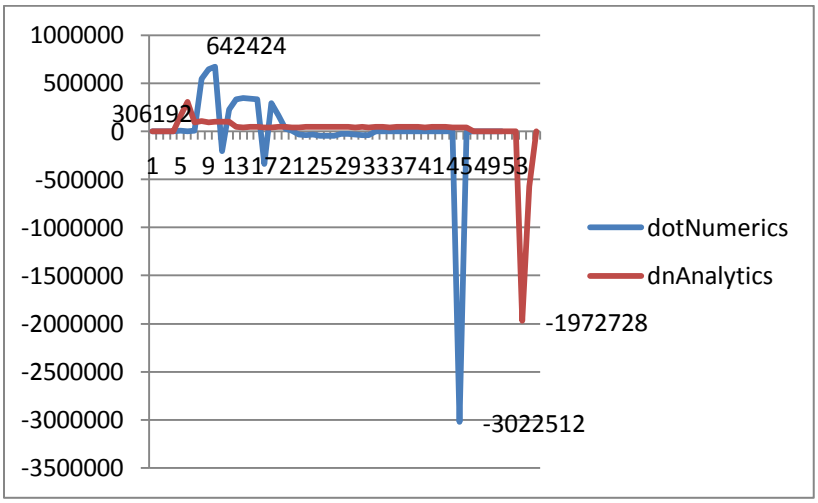

Gráfico 39

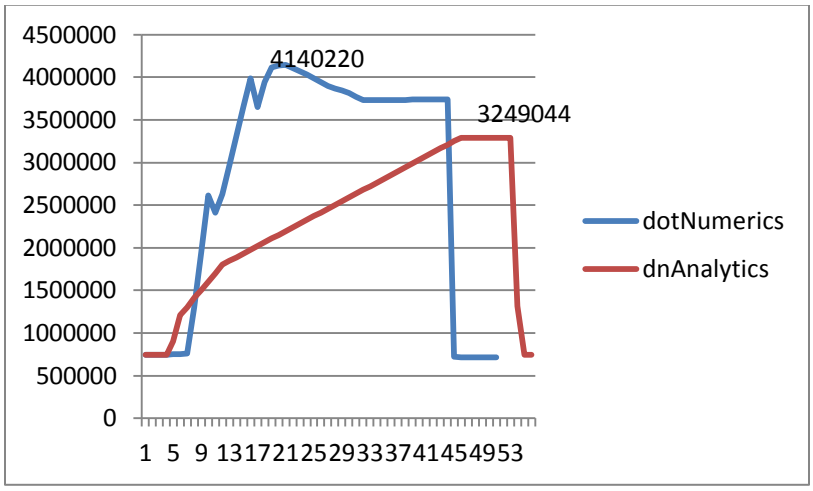

Gráfico 40

Se observa un crecimiento muy grande de *DotNumerics* produciendo una gran cantidad de consumo de RAM y un crecimiento más lineal de *dnAnalytics*, produciendo un consumo menos elevado de la memoria antes citada (ver gráficos 39 y 40)

En el siguiente apartado se pretende mostrar la visión global de ambas librerías para poder hacer un análisis más acertado de cada una de las librerías.

5.2.4. Visión global

Librería	50		500		1.000		3.000		5.000		Varios	
	CPU	RAM	CPU	RAM	CPU	RAM	CPU	RAM	CPU	RAM	CPU	RAM
dnAnalytics	1.32%	5.906	2.17%	14.216	5.49%	33.891	10.51%	306.057	29.08%	923.639	46.52%	1.536.054
DotNumerics	0.98%	5.418	2.43%	20.635	5.50%	63.648	14.65%	490.828	20.32%	1.557.260	21.99%	2.060.519

Tabla 8

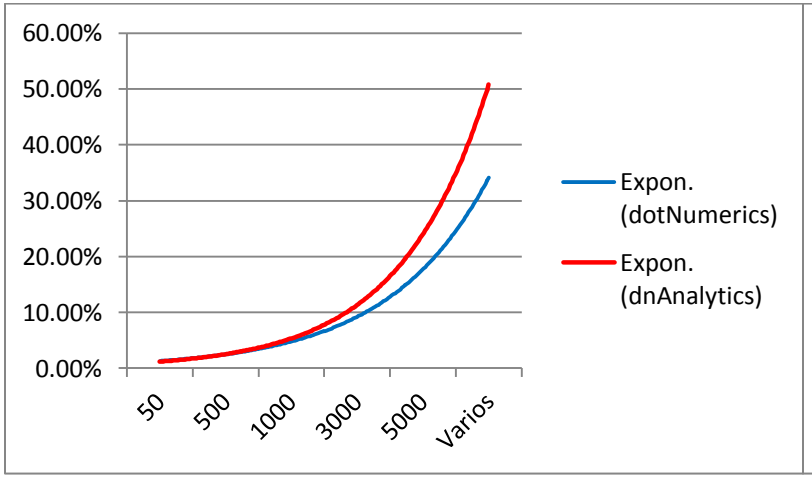

Gráfico 41

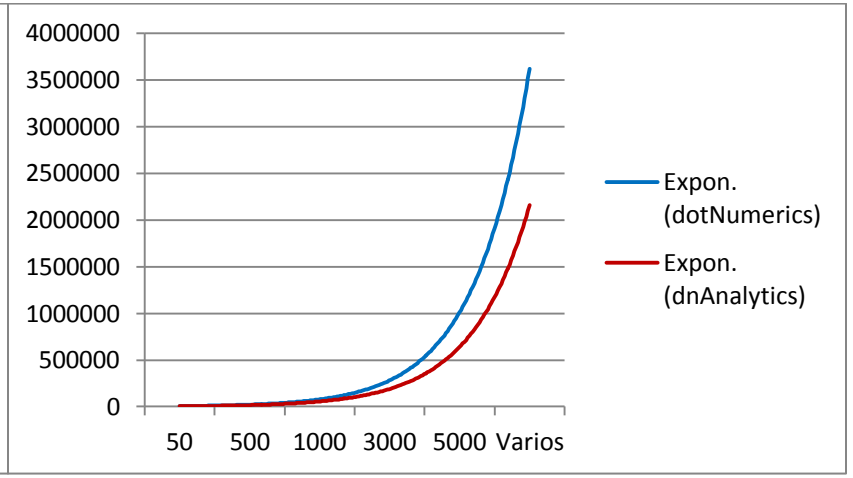

Gráfico 42

Al igual que en el primer análisis de este benchmarking a registros bajos se contempla un mejor rendimiento en la *dnAnalytics*, no obstante, a partir de los casos intermedios, *DotNumerics* penaliza en el consumo de RAM para conseguir una mayor rapidez y un menor uso de la CPU.

Una vez se han visto ambos análisis, se procederá a una conclusión. Estará dividida en tres partes, una por cada uno de los test realizados y la tercera para recoger los datos de las dos primeras y llegar a la conclusión final sobre que librería usar.

5.3. Conclusión

Recordemos los pesos asociados a las variables calculadas (ver gráfico 43):

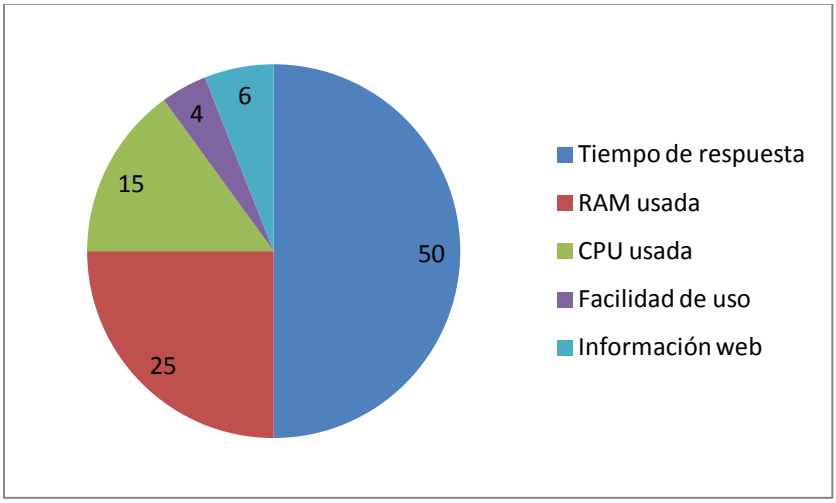

Gráfico 43

5.3.1. Thread

En la tabla 9, para cada una de las tres librerías, se le da un valor a las diferentes variables:

Variables/librería	Tiempo respuesta	CPU usada	RAM usada	Criterios subjetivos	
				Info. Web	*Facilidad uso*
algLib	0	--	--	5	6
DotNumerics	10	9	6	10	9
dnAnalytics	8	7	9	8	9
MathNet	6	--	--	9	10

Tabla 9

Calculemos los pesos para cada una de ellas:

- *AlgLib* = 0*0.5 + 5*0.06 + 6*0.04 = 0.54
- *DotNumerics* = 10*0.5 + 9*0.15 + 6*0.25 + 10*0.06 + 9*0.04 = 8.81
- *dnAnalytics* = 8*0.5 + 7*0.15 + 9*0.25 + 8*0.06 + 9*0.04 = 8.14
- *MathNet* = 6 * 0.5 + 9 * 0.06 + 10 * 0.04 = 3.94

Así pues, una vez asignados los valores de cada una de las diferentes variables se concluye que nuestra librería, en el caso de realizar un único *thread*, sería ***DotNumerics***.

AlgLib resulta descartada nada más comenzar el *benchmarking* por su alto tiempo de respuesta, *MathNet* por el análisis *Multi Thread* y *dnAnalytics* aun consumiendo menos RAM, sale perjudicada de los dos valores que más peso tienen (Tiempo de respuesta y CPU usada). Además siguiendo un criterio subjetivo bastante importante como es el de disponer de gran cantidad de información, *dnAnalytics* también sale perjudicada respecto la ganadora.

5.3.2. Multi-thread

A continuación se le dará un valor a cada uno de los pesos (ver tabla 10) para poder elegir la librería:

Variables/librería	Tiempo respuesta	CPU usada	RAM usada	Criterios subjetivos	
				Info. Web	*Facilidad uso*
algLib	0	--	--	5	6
DotNumerics	10	8	4	10	9
dnAnalytics	7	6	9	8	9
MathNet	5	--	--	9	10

Tabla 10

Calculemos los pesos para cada una de ellas:

- *AlgLib* = 0*0.5 + 5*0.06 + 6*0.04 = 0.54
- *DotNumerics* = 10*0.5 + 8*0.15 + 4*0.25 + 10*0.06 + 9*0.04 = 8.16
- *dnAnalytics* = 7*0.5 + 6*0.15 + 9*0.25 + 9*0.06 + 9*0.04 = 7.55
- *MathNet* = 3 * 0.5 + 9 * 0.06 + 10 * 0.04 = 2.44

Nuevamente gana ***DotNumerics***. Entrando de forma individual para cada una de las librerías, se observa como *algLib* sigue funcionando mal, *DotNumerics* mantiene un diez en tiempo de respuesta, se rebaja en uno el peso que se le da a la CPU usada y se rebajan dos puntos a la RAM usada por la gran diferencia con *dnAnalytics*. Finalmente baja un punto tanto en el tiempo de respuesta como en la CPU usada, mientras que se mantiene con el mismo valor en la RAM.

En último lugar comentar el caso de *MathNet*. Para múltiples *threads* resulta claramente perjudicada en el tiempo de respuesta, reduciendo de seis a tres el valor de éste.

5.3.3. Conclusión final

***DotNumerics* ha sido superior** en los dos test. En el análisis aislado ha conseguido 8.81 puntos mientras que su rival más directo, *dnAnalytics*, 8.14 y en el análisis *multi-thread* ésta última ha conseguido 7.55 mientras que la ganadora ha sumado 8.16 puntos.

6. Diseño del sistema

El proyecto está formado por tres partes claramente diferenciadas:

- La primera de ellas es el proyecto en Java UsoCpu, que servirá para recoger los datos de la CPU que posteriormente servirán para decidir que librería usar.
- La segunda parte la compone el código que testeará cada una de las librerías. A su vez este mismo está formado por dos tipos de proyecto, con un *Thread* o con Multi *Thread*.
- La tercera parte del proyecto es la codificación de la librería con la ganadora de la segunda parte.

6.1. Clases del diseño

A continuación se muestra el diseño de las clases para cada una de las partes anteriores:

6.1.1. Programa de captura de datos

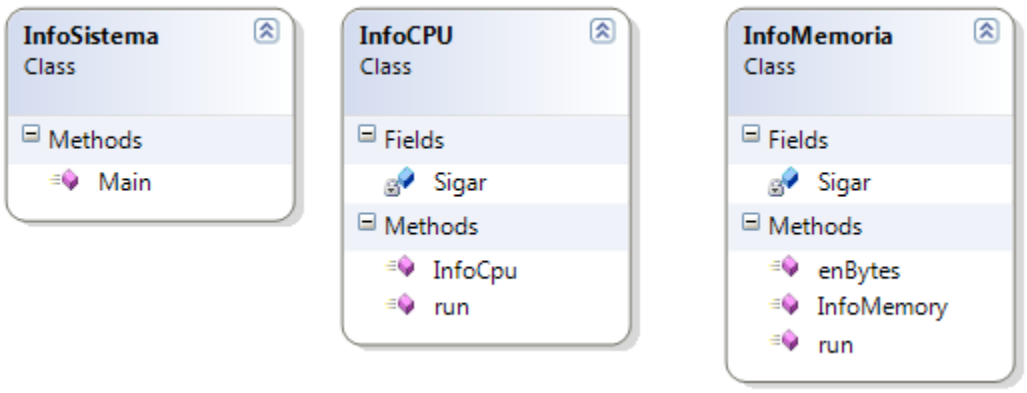

Figura 7

6.1.1.1. InfoSistema

- **Nombre de la clase:** InfoSistema.
- **Descripción:** Clase principal encargada de llamar a InfoCPU e InfoMemoria.
- **Cardinalidad:** Sin cardinalidad.
- **Atributos:** Sin atributos.

- **Operaciones:** Al ser un *main*, se encarga de arrancar los *Threads* además de crear instancias para llamar a los métodos tanto de la clase InfoCPU como de la clase InfoMemoria.

6.1.1.2. InfoCPU

- **Nombre de la clase:** InfoCPU.
- **Descripción:** Clase encargada de guardar en un fichero .csv los datos correspondientes al consumo de CPU. Irá volcando los datos al fichero hasta que se detenga manualmente el proceso. El intervalo de volcado de datos será de 750 milisegundos.
- **Cardinalidad:** Sin cardinalidad.
- **Atributos:**
    - Sigar: atributo de la clase sigar. Es preciso utilizar la librería externa sigar.jar para poder usar los métodos que ésta dispone para la obtención de los datos.
- **Operaciones:**
    - InfoCPU: Operación que gestiona los *threads* ya que InfoCPU () es un *extends* de ésta clase.
    - run: Operación encargada de ejecutar todo el proceso cada vez que el *thread* le da pasó a esta clase. Aquí se creará el fichero .csv que contendrá los datos siguientes: El número total de núcleos que compone nuestra máquina y el % uso de cada uno.

6.1.1.3. InfoMemoria

- **Nombre de la clase:** InfoMemoria.
- **Descripción:** Clase encargada de recoger los datos de la memoria del procesador en intervalos de 750 milisegundos. Los datos que se obtendrán serán mediante la librería sigar.jar. Éstos son:
    - *RAM:* El método getRam () se obtendrá la memoria RAM que tiene la CPU.
    - *RAM Delta:* Mediante una resta de la memoria RAM usada en el intervalo actual y en el anterior, se formará la columna RAM Delta que posteriormente será útil para el *benchmarking*.

- *RAM Total:* Toda la memoria que forma la CPU.
- *RAM Used:* Mediante el método getUsed () se podrá recoger los datos de la memoria RAM que se está usando en ese instante de tiempo.
- *RAM Available:* Mediante el método getFree () se puede obtener el total de memoria RAM disponible.
- *SWAP Total:* Para obtener instancias de la memoria *swap* se ha de llamar al método getSwap (). Esta columna será rellenada mediante el método getMem () para obtener la memoria *swap* total que tiene la máquina.
- *SWAP Used:* Con el método getUsed () se podrá obtener la memoria *swap* usada.
- *SWAP Available:* Gracias al método getFree () se podrán obtener los valores disponibles de la memoria *swap*.

- **Atributos:**
  - Sigar: atributo de la clase sigar. Es preciso utilizar la librería externa sigar.jar para poder usar los métodos que ésta dispone.
- **Operaciones:**
  - InfoMemory: Operación necesaria ya que esta clase es un *extends* de esta clase.
  - run: Operación encargada de ejecutar todo el proceso cada vez que el *thread* le da pasó a éste. Como se ha explicado en la descripción, mediante los distintos métodos de la librería sigar y los objetos sigar.getMem () y sigar.getSwap (), se podrán obtener las columnas del archivo .csv.
  - enBytes (): Operación privada encargada de transformar los datos obtenidos a Bytes.

6.1.2. Programa testeo de tiempos

6.1.2.1. Thread

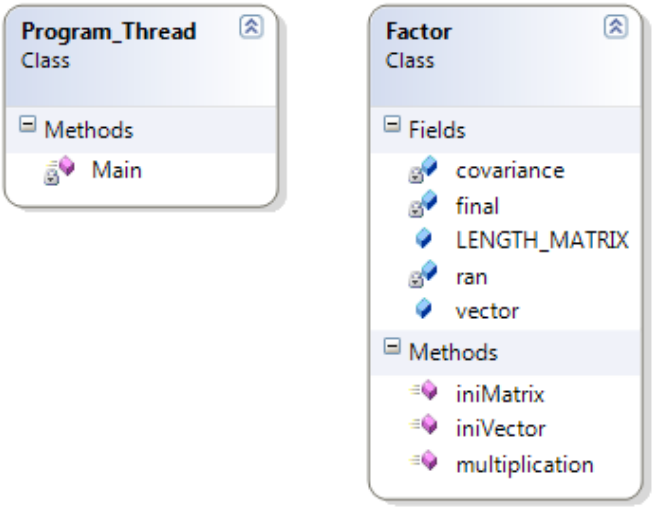

Figura 8

Program Thread

- **Nombre de la clase:** Program _Thread.
- **Descripción:** Clase principal (*main*) encargada de realizar el *Factor Risk* para cada una de las librerías. Esta clase también calculará el tiempo de respuesta.
- **Cardinalidad:** Sin cardinalidad.
- **Atributos:** Sin atributos.
- **Operaciones:** Al ser un main únicamente crea instancias para ejecutar los métodos de la clase Factor.

Factor

- **Nombre de la clase:** Factor.
- **Descripción:** Clase encargada de ejecutar las inicializaciones de los diferentes objetos que formarán parte del proyecto.

- **Cardinalidad:** Sin cardinalidad.

- **Atributos:** Todos los atributos que esta clase contiene son los relativos a la librería usada para ese proyecto, a excepción del entero *LENGHT_MATRIX*, que es común a todos. En el caso de la figura 8 se ha usado la *DotNumerics*. A continuación se hace una descripción de cada uno:

    - Covariance: matriz de tipo *Matrix*.

    - Final: matriz de tipo *Matrix*.

    - LENGHT_MATRIX: entero de tipo genérico que indicará la longitud que ha de tener tanto las matrices como el vector a tratar.

    - Ran: atributo de tipo *Random* de *DotNumerics* que nos genera números aleatorios. Esto se usará para inicializar tanto la matriz como el vector.

    - Vector: vector de tipo *Vector* de la librería *DotNumerics.*

- **Operaciones:** Son las siguientes:

    - iniMatrix: método que permite la generación aleatoria de una matriz de tamaño LENGHT_MATRIX.

    - iniVector: vector de números aleatorios de tamaño LENGHT_MATRIX.

    - Multiplication (): se hará la creación de los objetos Matrix y Vector, la inicialización de cada uno de éstos y realizará las operaciones para el cálculo del *Factor Risk.*

6.1.2.2. Multi Thread

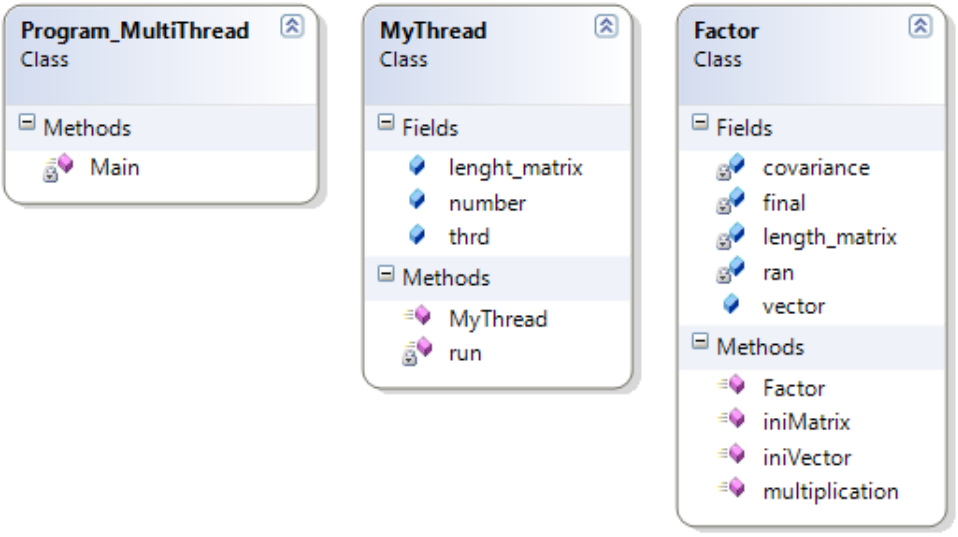

Figura 9

Program Multithread

- **Nombre de la clase:** Program _Multithread
- **Descripción:** Clase encargada (*main*) de inicializar los diez *Threads* que tendrá nuestro test. En esta clase también se hará el cálculo del tiempo de respuesta.
- **Cardinalidad:** Sin cardinalidad.
- **Atributos:** Sin atributos.
- **Operaciones:** Al ser un main únicamente crea instancias para ejecutar los métodos de la clase *MyThread*.

MyThread

- **Nombre de la clase:** MyThread
- **Descripción:** Clase encargada de lanzar cada uno de los procesos que ha de realizar cada *Thread*
- **Cardinalidad:** Sin cardinalidad.

- **Atributos:**

    - Number: entero que el único objetivo es identificar el *Thread*.

    - Thrd: atributo de tipo genérico *Thread*. Es necesario este atributo para iniciar el método de arranque *Start* () para cada uno de los *threads*.

    - Lenght_matrix: atributo de tipo entero que nos da información sobre la longitud que ha de tener tanto la matriz como el vector.

- **Operaciones:** Esta clase contiene dos métodos:

    - MyThread: constructora de la clase que recibe como parámetros un entero llamado *lenght* y otro llamado num. El primero se usa para asignar el tamaño de la matriz y el vector y el segundo para identificar el *thread*.

      En la constructora también se lanza aquello que ha de realizar cada uno de los *Threads* mediante el método comentado *run*.

    - Run: método que permite llamar a la clase Factor para que ésta realice los objetivos de estudio de este *test*.

Factor

- **Nombre de la clase:** Factor.

- **Descripción:** Encargada de hacer las inicializaciones de los diferentes objetos que formarán parte en el método *multiplication* () para calcular el *Factor Risk*.

- **Cardinalidad:** Sin cardinalidad.

- **Atributos:**

    - Covariance: matriz de tipo *Matrix* (tipo de la librería *DotNumerics* exclusivo para matrices).

    - Final: al igual que el caso anterior, matriz de tipo *Matrix*.

- LENGHT_MATRIX = entero de tipo genérico que indicará la longitud que ha de tener tanto las matrices como el vector a tratar.

- Ran: atributo de tipo *Random* de *DotNumerics* que nos genera números aleatorios. Esto se usará para inicializar tanto la matriz como el vector. Vector: vector de tipo *Vector* de la librería *DotNumerics.*

- **Operaciones:** Las operaciones que contiene esta clase son muy parecidas a la de un único *thread*, salvo una excepción que se verá al inicio.

  - Factor: constructora de la clase que únicamente sirve para parametrizar el valor Lenght_matrix con el valor que proviene de MyThread.

  - iniMatrix: método que permite la generación aleatoria de una matriz de tamaño LENGHT_MATRIX para las filas y las columnas.

  - iniVector: método que permite la generación aleatoria de un vector de tamaño LENGHT_MATRIX.

  - Multiplication: método que contendrá la creación de los objetos Matrix y Vector, llamará para la inicialización de cada uno de éstos y realizará las operaciones pertinentes para el cálculo del *Factor Risk.*

6.1.3. Librería

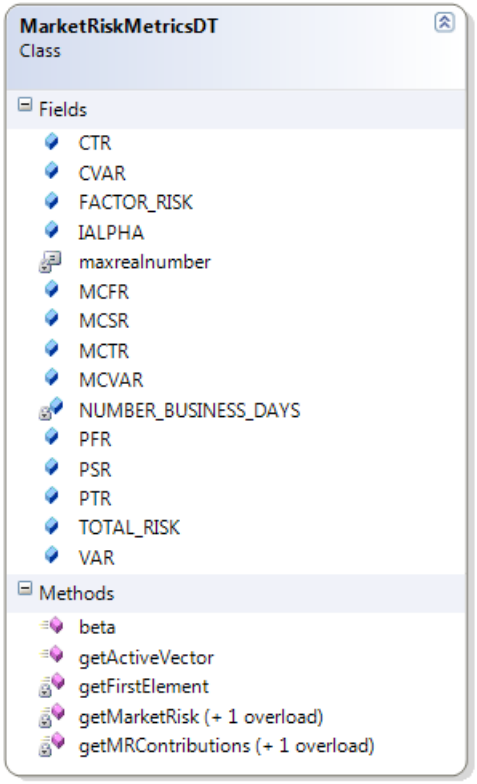

Figura 10

La librería tendrá la estructura de la figura 10. Se puede observar en dos métodos lo siguiente: (+1 overload[16]). La sobrecarga de métodos en programación orientada a objetos sirve para que dadas dos o más funciones con el mismo nombre, pero con distintos parámetros de entrada, se puedan llamar y ejecutar sus funcionalidades sin problema alguno.

Por un lado tendremos el método público getMarketRisk y el privado con el mismo nombre y por otro lado tenemos el público getMRContributions con su respectivo privado.

Los métodos públicos recibirán objetos de tipo estándar y éstos a su vez serán transformados a objetos de tipo *DotNumerics* para que los métodos privados puedan realizar las operaciones con estos objetos.

[16] Sobrecarga de métodos en Programación Orientada a Objetos (POO)

Por tanto si en un momento dado se decide cambiar la librería externa no representaría ninguna complicación para el encargado de utilizarla ya que éstos trabajarán únicamente con las públicas y éstas últimas ya se encargarían de hacer lo necesario para que las privadas puedan funcionar sin problema. No obstante el desarrollador tendría que revisar la documentación de la nueva librería ya que las operaciones que se produzcan en las privadas, se han de adaptar a ésta.

A continuación se entrará a definir la librería:

- **Nombre de la clase:** MarketRiskMetricsDT
- **Descripción:** Única clase del proyecto y como tal será la encargada de recibir todos los *inputs* comentados en el apartado 3.4.1 y transformarlos en los *outputs* del apartado 3.4.2 de esta misma documentación.
- **Cardinalidad:** Sin cardinalidad.
- **Atributos:**
    - CTR: entero. Éste y los siguientes enteros servirán para identificar posiciones en un vector. De esta manera será más sencillo situarse correctamente en él y no machacar valores. CTR contendrá el valor de *Contribution to Total Risk*.
    - CVAR: entero. *Contribution to VaR.*
    - FACTOR_RISK: entero. *Factor Risk.*
    - IALPHA: entero. *Implied Alpha.*
    - MCFR: entero. *Marginal Contribution to Factor Risk.*
    - MCSR: entero. *Marginal Contribution to Specific Risk.*
    - MCTR: entero. *Marginal Contribution to Total Risk.*
    - MCVAR: entero. *Marginal Contribution to VaR.*
    - PFR: entero. % *Factor Risk.*

- PSR: entero. % *Specific Risk.*
- TOTAL_RISK: entero. *Total Risk* del *Portfolio.*
- VAR: entero. *Value at Risk*.
- Maxrealnumber: Double. Se usará para hacer el cálculo de la distribución normal inversa para el proceso del cálculo del *VaR*, *MCVAR* y *CVAR.*
- NUMBER_BUSINESS_DAYS: entero. Indicará el número de días laborales por año. Se usará para el cálculo de los mismos casos que la constante Maxrealnumber.

- **Operaciones:**

  - Beta: Calculará la beta. Recibe como parámetro el Double *Portfolio Total Risk,* el *Benchmark Total Risk* y el *Active Total Risk.*
  - getActiveVector: operación encargada de retornar una tabla de *Hash*. Como tabla de *Hash* se entiende como una estructura de datos que dada una clave, ésta tiene asociada un valor. Siempre que se habla de esta estructura, es importante decir que las claves no pueden ser repetidas, mientras que los valores sí. A continuación se puede ver dos ejemplos (ver figura 11), uno de una implementación correcta y otro que no lo es.

Clave	Valor
X	1
Y	2
Z	1

Clave	Valor
X	1
Y	2
X	3

Figura 11

La primera tabla sí es una implementación correcta ya que aún repitiendo valor, ninguna de las tres claves se repite, mientras que la segunda no se repite ningún valor pero sí se repite una clave.

Así pues una vez se tiene clara la definición de tablas de hash se pasará a explicar exactamente que hace el método que se está tratando ahora.

A la hora de declarar de que tipo es la función se tendrá que especificar lo siguiente: *System.Collections.Hashtable*, retornando así una tabla de *hash*.

Tendrá una serie de parámetros de entrada, cuatro vectores, dos de ellos *Strings* y dos de tipo *Double*. Los vectores de *Strings* harán de claves en nuestra tabla, mientras que los *Double* serán el valor. Como se puede ver, todos son tipos básicos, es decir en este método no se utilizará la librería escogida en el *benchmarking*.

Los vectores *Strings* (clave) serán los siguientes:

- *Portfolio Factor Vector*. Vector de Strings que contendrá todos los factores de riesgo del *Portfolio* en cuestión.
- *Benchmark Factor Vector.* Vector de Strings que, al igual que el anterior, contendrá todos los valores de riesgo del *Benchmark.*

Los vectores *Double* (valor) serán los siguientes:

- *Portfolio Exposure Vector*. Valores Double asociados para cada uno de los riesgos anteriores.
- *Benchmark Exposure Vector.* Double asociado para cada uno de las claves del *Benchmark Factor Vector.*

La tabla de hash resultante ha de tener todos los factores de riesgo del *Portfolio* además de los que, sin coincidir con el del *Portfolio*, sean nuevos en el *Benchmark Exposure Vector.*

En el caso de que haya una o varias claves de ambos vectores que coincidan, se cogerá el valor de la clave del *Benchmark* y se le restará al mismo del *Portfolio*. Veamos el siguiente ejemplo:

Portfolio Factor Vector

F1	*F4*	*F3*	*F7*	*F2*

Portfolio Exposure Vector

5	12	8	20	7

Benchmark Factor Vector

F6	*F7*	*F1*	*F8*

Benchmark Exposure Vector

15	3	9	25

Una vez se tienen definidos los vectores ejemplos se describirán los pasos que sigue nuestro código.

- Poner en la tabla de hash todas las claves que aparecen en el *Portfolio Factor Vector* y asociar los valores a éstas cada una de las posiciones del vector *Portfolio Exposure vector.*

F1	*F4*	*F3*	*F7*	*F2*
5	12	8	20	7

- Una vez se tienen agregados en la tabla de hash todos los par clave/valor del *Portfolio*, se ha de añadir los del *Benchmark* tal y como se ha comentado antes. En caso de que un factor de este vector coincida con uno que se encuentra en la tabla de hash se cogerá el valor de éste y se restará al existente de la tabla de hash y en el caso de que no esté, se añadirá a la tabla negando su valor.

F1	*F4*	*F3*	*F7*	*F2*	***F6***	***F8***
-4	12	8	**17**	7	**-15**	**-25**

En rojo están los cambios observados después de insertar el *Benchmark Factor Vector*. El factor F1 contenía como valor 5, posteriormente se ha de restar el valor 9 perteneciente al último vector.

Con el factor F7 ocurre lo mismo, del 20 inicial que contiene el vector del *Portfolio*, se pasa a los 17 al restar el valor 3 del *Benchmark.* Finalmente los factores F6 y F8 se han añadido al final de la tabla de hash ya que no coinciden con ninguno de los ya introducidos negando el valor inicial.

- getMarketRisk (pública): Operación encargada de transformar los tipos básicos en los tipos específicos de nuestra librería. Tanto ésta como la pública getMRContributions serán claves para entender la encapsulación en el proyecto. Para realizar esta conversión únicamente nos hemos de asegurar que los tipos de la librería externa contienen un método para inicializarlos mediante un tipo básico. Por supuesto *DotNumerics* los tiene, así que simplemente con hacer la siguiente sentencia:

  ```
  Matrix covarMx = new Matrix (covarM);
  ```

  Transformará una matriz de tipo básico (Double [,] covarM) a otra de tipo específico de la librería (Matrix). Para el caso del vector es exactamente el mismo. Una vez se han hecho las transformaciones, únicamente quedará llamar a la función privada pasando tanto los tipos específicos, como los *Double* para realizar todas las operaciones necesarias.

- getMarketRisk (privada): Realiza los cálculos del *Factor Risk*, *VaR* y *Total Risk*. Recibe los tipos específicos que se han transformado en la pública y guarda para cada una de las posiciones del vector resultado, un Double asociado a cada uno de los tres valores de riesgo. La codificación no se pondrá puesto que se adjuntará en un CD junto con la memoria.

  Al seguir las fórmulas surge una duda y es el cálculo de la inversa de la distribución normal. Se ha añadido una clase que la empresa disponía para hacer el cálculo que posteriormente será usado asignado en el *input* nivel de confianza.

- getMRContributions (pública): La razón que exista es para transformar los tipos genéricos en los tipos específicos de la clase de forma transparente. Recibirá una matriz y dos vectores, además de cuatro *Double*.

  Como se usarán valores calculados en la función anterior también recibiremos el vector resultado del primer apartado como parámetro de entrada para evitar llamar a la

función getMarketRisk para obtener los valores cada vez que se haga la llamada a la función que se está tratando.

- getMRContributions (privada): Operación que calcula todos los marginales. Recibe una matriz de tipo Matrix, dos vectores de tipo Vector, 5 *Doubles* y el vector resultado del getMarketRisk.

  Como en el caso de la privada anterior, se va guardando cada uno de los resultados en un vector de *Doubles*. Dicho vector tendrá diez posiciones, a diferencia de los tres que tenía la primera de las privadas.

Una vez analizados todos los métodos de la librería FinLib.dll se pretende explicar cómo se ha de crear una librería en un proyecto con el Microsoft Visual Studio 2010. Los pasos a seguir son:

- Abrir Microsoft Visual Studio 2010 y creamos un nuevo proyecto librería (.DLL).
- A continuación será crear la clase. El código de ésta como se ha comentado con anterioridad, se incluirá en un CD cuando se entregue la memoria del proyecto.
- Una vez creada la clase, se selecciona con el botón derecho del ratón sobre el proyecto y vamos a la opción *Build* generando así el fichero librería.

6.2. Diagrama del sistema

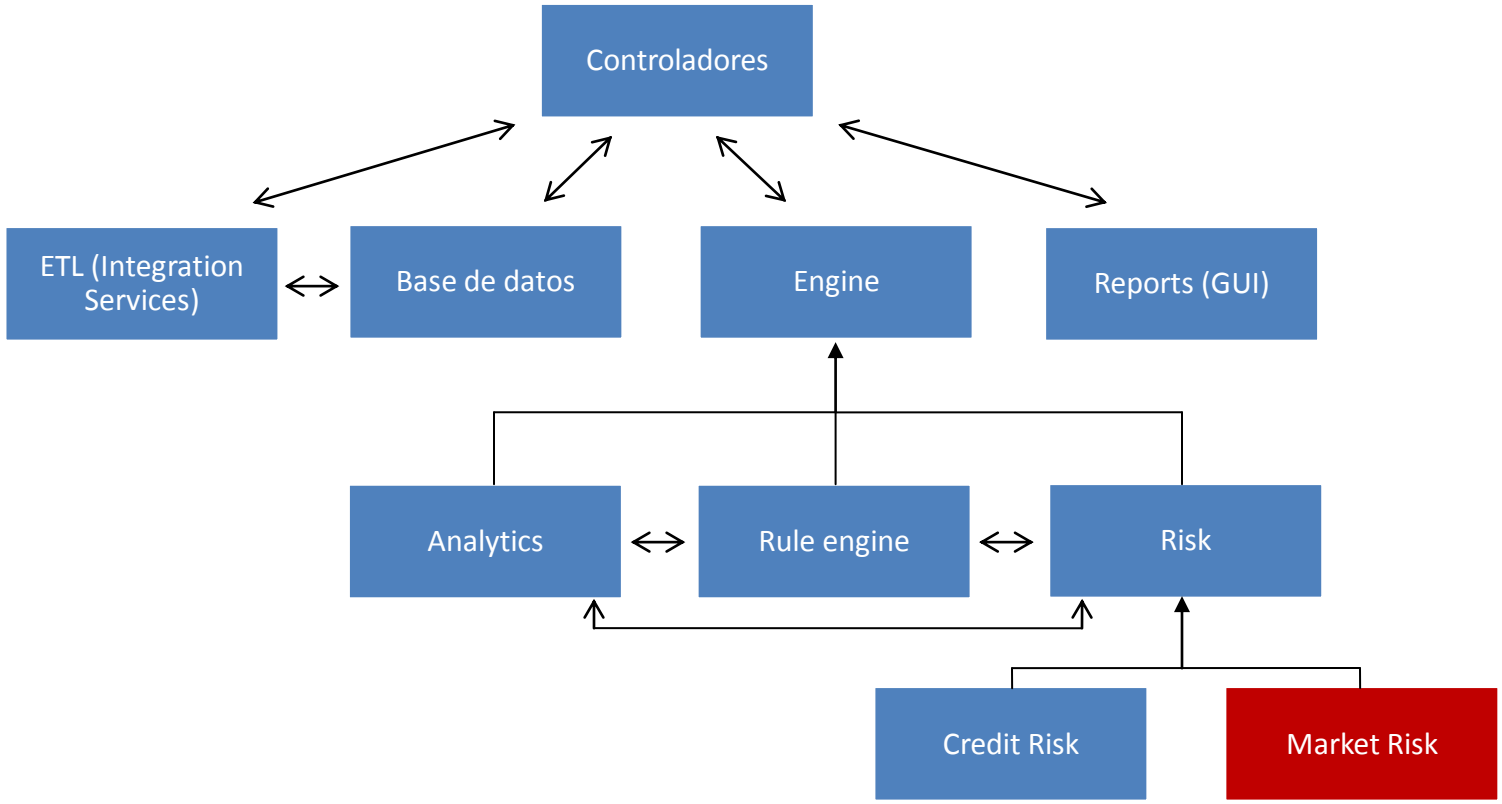

Figura 12

6.2.1. Módulos

El sistema PATONE está formado por múltiples secciones (ver figura 12):

- Controladores: Encargados de gestionar la comunicación entre los cuatro primeros niveles del sistema. La razón de la existencia de los controladores es que en el caso que se produzca un cambio en el diseño del sistema estas variaciones no produzcan una modificación drástica del código.

  Permitirán mantener cada uno de los módulos del sistema totalmente independiente de los demás haciendo así más fáciles las posibles mejoras/modificaciones del futuro.

- ETL (*Extract Transformation Load)* se encarga de cargar los datos que provienen de distintos proveedores en la base de datos. El paso por los controladores muchas veces es vital para realizar determinadas operaciones. Una de ellas es la que se desglosa a continuación:

  - *ETL* recibe una serie de datos de los proveedores.
  - La *ETL* envía una petición para el cálculo del VAR al controlador.
  - El controlador llama al módulo *Engine* y éste se encarga de llamar al módulo correcto (*Risk* → *Market Risk Engine*)
  - El módulo anterior retornaría los valores del cálculo al controlador.
  - El controlador enviaría el resultado a la *ETL.*
  - Finalmente ésta última volcará los datos a la base de datos.

- Base de datos: Este módulo contiene una estructura que modela el sistema financiero actual. Aquí se tendrán tablas de transacciones, fondos, carteras y *Securities*.

- *Engine*: Parte del sistema donde se producen todos los cálculos. Está formada por *Analytics*, *Aggregation Grouping* y la parte del *Risk*, que es donde nosotros entraremos.

Estos cálculos se volcarán en unos reportes que serán analizados cuidadosamente por un *Portfolio Manager* para dar unas conclusiones y decantarse por ejemplo si ejecutar una opción de compra para una *Security.*

- *Reports* (GUI): Módulo que permite hacer minería de datos. Se entiende por minería de datos a la obtención de información con métodos no triviales, es decir, se puede tener mucha información en la base de datos y en los reportes, pero si estos no son analizados para sacar información "oculta" no serviría de nada.

- Analytics: Sub-módulo de *Engine.* Contendrá una serie de variables que nos explicarán el comportamiento de cada instrumento financiero. Estas variables se calculan mediante diferentes modelos matemáticos que dispone la propia empresa o de externos como *Barclays.*

- *Rule Engine*: Sub-módulo de *Engine.* Implementa las reglas de negocio. Hace funciones como *Aggregation* o el cálculo del límite. Además se realizan una serie de técnicas *OLAP*[17] para hacer resúmenes. Se entiende por estas técnicas todas aquellas que sirven para hacer más fácil y rápido la consulta de grandes cantidades de datos.

- Risk: Este último también es un sub-módulo del *Engine* y es aquí donde el proyecto será alojado. Está formado por nuestra librería y en un futuro contendrá el módulo de *Credit Risk.*

- Credit Risk: Este módulo aún no está implementado. Cuando esté, se harán los cálculos del riesgo de crédito que la empresa crea conveniente.

- Market Risk: Este es el módulo nuevo que se agregará en el proyecto. Simplemente consistirá en la librería que hemos creado y ya el propio *Engine* hará lo que tenga que hacer con todos los datos obtenidos en esta librería.

6.2.2. Comunicación entre los módulos[18]

Una de las partes esenciales de todo diagrama de un sistema es explicar ¿Cómo es la comunicación entre cada uno de los módulos? Primeramente clasificaremos cada uno de los módulos por niveles.

[17] *On-Line Analytical Processing*
[18] Se recomienda seguir la figura 12 página 80 para facilitar el seguimiento

- Nivel 1: Controladores.
- Nivel 2: *ETL*, Base de datos, *Engine*, Reportes.
- Nivel 3: *Analytics, Rule Engine, Risk*.
- Nivel 4: *Credit Risk, Market Risk*.

Lo primero a comentar será la comunicación entre distintos niveles:

- Nivel 1 – Nivel 2: Comunicación bidireccional. Necesario para realizar muchas operaciones como la siguiente: La *ETL* recibe datos de un proveedor, manda petición al controlador, éste manda petición al *Engine* que a su vez manda una orden al *Market Risk Engine* para obtener el VaR. Este cálculo hace el camino inverso hasta la *ETL* donde finalmente el valor resultante será volcado a la base de datos.
- Nivel 2 – Nivel 3: Realmente no hay comunicación implícita. Se trata de una jerarquía donde los tres módulos del nivel 3 provienen del módulo *Engine* alojado en el nivel 2. Los otros módulos de este último nivel, no se comunican de ninguna manera con los tres del nivel inferior.
- Nivel 3 – Nivel 4: Al igual que en el caso anterior, no hay comunicación implícita entre ellos. Únicamente se trata una herencia de un módulo del nivel 3 (*Risk*) que influye a los dos módulos del nivel más bajo de todos.

La comunicación entre módulos del mismo nivel es la siguiente[19]:

- Nivel 1: Únicamente encontramos un único módulo, por tanto no hay comunicación.
- Nivel 2: Comunicación que se ve afectada por la implantación del módulo de controladores.
    - *ETL* – Base de datos: Bidireccional. Básico para que la base de datos obtenga todo lo que la ETL obtiene de los proveedores y también crucial para que la *ETL* pueda hacer consultas puntuales en la base de datos para poder realizar algunas de las peticiones que se le puedan mandar a través de los controladores al módulo *Market Risk Engine.*

[19] Se recomienda seguir la figura 12 página 80 para facilitar el seguimiento

- Base de datos – *Engine:* No hay comunicación entre ellos. Si se desea realizar esta operación, nos veremos obligados a pasar por los controladores.

- *Engine* – Reportes: Al igual que el caso anterior, no hay comunicación entre estos dos módulos a no ser que se pase por los controladores.

- Nivel 3: Comunicación bidireccional entre todos y cada uno de los módulos. Esta comunicación será esencial para que por ejemplo con los datos obtenidos en diferentes consultas de VaR, en el módulo *Rule Engine* se hagan técnicas OLAP para obtener diferentes de estos valores.

- Nivel 4: No se prevé comunicación cuando el módulo *Credit* Risk esté hecho ya que hacen cálculos completamente diferentes.

6.3. Casos de uso

Como todo sistema también existirán unos posibles casos de uso donde nuestra librería intervendrá.

- Abrir un *Portfolio*: La Compañía dispone de una aplicación web (*PAT One Reporter*) de *reporting* que permite al *Portfolio manager* visualizar su cartera de inversión. Entre varias funcionalidades permite la exportación a Excel:

	MV Gross Base	MV Wgt.	CMFR	TMR	VAR	MCTR	MCSR	MCFR	MCVAR	CVAR	PTR	PFR	IALPHA
Total Fund	,978,902,170	100.000%	16.57	16.66	7.72	16.66	1.72	16.57	7.72	7.72	16.66	16.57	
GS CALTEX CORP 5.5 04/24/2017	,936,721	.300%	15.22	39.53	18.32	.00	.42	.10	.00	.00	.00	.00	.07
GLOBO COMUNICACAO E PARTI 7.25 04/26/2022	0,965,083	.704%	41.25	47.63	22.07	.01	.58	.37	.00	.00	.00	.01	.13
TNK-BP FINANCE SA 6.625 03/20/2017	,991,250	.235%	33.80	55.92	25.91	.00	.37	.30	.00	.00	.00	.01	.09
ALFA DIVERSIFIED PAYMENT 4.81875 03/15/2012	,800,253	.060%	10.92	23.17	10.74	.00	.05	.09	.00	.00	.00	.00	.01
WOORI BANK 6.208 05/02/2037	,183,022	.174%	11.87	60.34	27.96	.00	.32	.08	.00	.00	.00	.00	.09
PETROLEUM CO OF TRINIDAD 6 05/08/2022	4,074,514	.472%	41.61	51.22	23.74	.00	.32	.32	.00	.00	.00	.01	.10
INTERGAS FINANCE BV 6.375 05/14/17	,255,000	.311%	34.70	57.41	26.61	.00	.39	.10	.00	.00	.00	.00	.11
PROVINCE OF BUENOS AIRES 9.375 09/14/2018	77,500	.009%	22.83	62.63	29.02	.00	.15	.16	.00	.00	.00	.01	.01
PETROLEOS DE VENEZUELA SA 5.25 04/12/2017	,649,767	.055%	42.39	73.16	33.90	.00	.34	.32	.00	.00	.00	.01	.03
TRANSCAPITALINVEST LTD FO 5.67 03/05/2014	,573,475	.254%	26.30	38.55	17.86	.00	.27	.23	.00	.00	.00	.00	.04

Figura 13

A parte de otros campos descriptivos y de análisis, la aplicación web mostrará el resultado de las llamadas a nuestra librería (ver figura 14 columnas azules y grises).

La interacción que se hará en este caso de uso con los distintos módulos es la siguiente:

- El usuario hace una petición para abrir su cartera a la aplicación.
- La aplicación llama al controlador indicando la cartera a abrir.
- El controlador carga los datos de la cartera de la base de datos.
- El controlador llama a FinLib para el cálculo de las métricas de riesgo.
- FinLib devuelva las métricas de riesgo.
- El controlador devuelve los resultados a la aplicación web.
- La aplicación web dará formato a los resultados y los devuelve al usuario.

- Risk Decomposition: El objetivo de este caso de uso es descomponer el *Total Risk*, *Factor Risk* y el *Value at Risk* por factores. Observemos como se interacciona con el sistema:
  - Todos los casos anteriores para abrir un Portfolio.
  - A continuación el usuario hace una petición para abrir el reporte de Risk Decomposition.
  - La aplicación llama al controlador indicando de que cartera quiere obtener el Risk Decomposition.
  - El controlador indica al motor de agregación que agregue las Securities del Portfolio por factores.
  - El controlador llama a FinLib para calcular el Total Risk de cada una de las agrupaciones del *Portfolio.*
  - FinLib devuelve los cálculos.
  - El controlador devuelve los resultados a la aplicación web.
  - La aplicación web dará formato a los resultados y los devuelve al usuario.

  En la figura 15 se muestra un ejemplo que genera la PATOne Reporter Tool para este segundo caso de uso.

Factor Group	Total Risk	Benchmark Risk	Active Risk	%Active Risk
Total Risk	16.664	25.799	16.521	100.00 %
Common Factor Risk	16.572	18.416	2.567	15.54 %
Currency Risk	1.858	2.604	0.889	5.38 %
Developed Currency Risk	1.858	2.604	0.889	5.38 %
EM Currency Risk				

Figura 14

- What if Scenarios: Este caso de uso se da cuando el usuario modifica uno de los datos obtenidos del caso de uso de abrir un Portfolio. La interacción que se produce es:

    - Todos los casos de uso de abrir un Portfolio.
    - El usuario indica en el PATONE Reporter Tool que quiere modificar.
    - El sistema muestra un Pop-Up (ver figura 16) donde se muestra el valor original y un Label para que el usuario introduzca el nuevo valor.
    - El sistema almacenará el nuevo valor.
    - El controlador llama a FinLib para el cálculo de las métricas de riesgo con el valor que se ha indicado antes.
    - FinLib devuelva las métricas de riesgo.
    - El controlador devuelve los resultados a la aplicación web.
    - La aplicación web dará formato a los resultados y los devuelve al usuario.

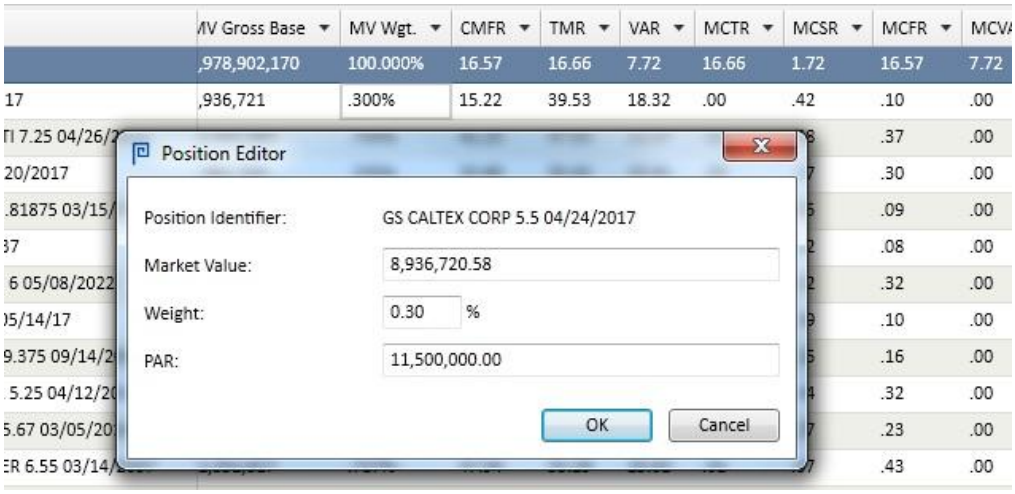

Figura 15

- **Group by:** Se agrupan las Securities por uno o varios campos que selecciona el usuario. Por ejemplo si se decide agrupar por el factor COUNTRY, tendremos todas las *securitites* del COUNTRY con valor USA juntas, las de EUR juntas y así para todos los valores que pueda tener el factor COUNTRY. Veamos cómo se interacciona con el sistema:

    - Casos de uso de abrir un *Portfolio*.
    - El usuario selecciona un campo o dos campos para hacer *Group By* (ver figura 17). En el caso ejemplo, se agrupa por *Risk Ctrv.*
    - El controlador llama al motor de agregación para todos los campos de la vista.
    - Para cada grupo el controlador llama FinLib
    - Para cada *Security* el controlado llama a FinLib para las MCTR
    - El controlador devuelve los resultados a la aplicación web.
    - La aplicación web dará formato a los resultados y los devuelve al usuario.

	PFR	IAL
Total Fund	16.57	
Oil&Gas	1.24	
Telecommunications	1.51	
Diversified Finan Serv	1.24	
Banks	1.35	
Gas	16	
Regional(state/provnc)	00	
Sovereign	9.02	
Retail	01	
Mining	99	
Electric	62	
Transportation	10	
Aerospace/Defense	13	

Figura 16

6.4. Testeando la librería

Como toda generación de un código, este necesita de un testeo real para saber si funciona y se comporta como se espera. En las siguientes líneas se describe como se ha testeado la librería.

Mediante un *Portfolio* real obtenido de la base de datos con la herramienta *PATOne Reporter Tool*, se compara cada una de las posiciones todos los valores objetivos con los datos obtenidos de nuestra librería.

Debido a que la comparación de cada una de las posiciones resultaría muy tediosa, se ha automatizado el proceso de tal manera que se creará un fichero .csv donde se irán volcando todas las parejas de valores, es decir el valor del reporte con el valor de nuestra librería seguido de una resta entre ambas, de tal manera que si es cero o prácticamente cero, el valor generado por la librería será el correcto.

Con esto ya será suficiente para verificar que los valores de riesgo calculados cuadran con los del reporte. No obstante se debe verificar que el *Active Vector* también lo hace bien.

En el reporte no aparece este vector, pero si los valores de riesgo asociados a éste. Así pues para saber si la librería hace bien este proceso bastará con llamar a la función getActiveVector, obtener el vector y para cada una de las posiciones, calcular el valor de riesgo asociado.

6.5. Uso de la librería en un proyecto externo

Si bien esto no es una clase propiamente dicha, si sería interesante añadir este apartado aquí ya que viene ligado directamente con lo anterior. Se pretende hacer un pequeño tutorial para añadir una referencia a la librería. Para ello se han de seguir los siguientes pasos:

- Abrir Microsoft Visual Studio 2010 y crear un nuevo proyecto.

- Clic derecho sobre el apartado *References* → *Add References.* En la ventana que se abrirá ir a la pestaña *Browse* e ir al directorio *bin/debug/* del proyecto. Aquí se encontrará el archivo librería generado en el *Build.*

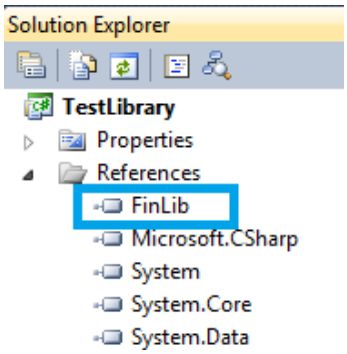

Figura 17

En la figura 18 se ve la librería *FinLib*.dll añadida para poder utilizar los métodos que ahí se han codificado. Las referencias restantes son las comunes de todo proyecto que se inicie desde cero.

- No obstante no basta con añadir la referencia en el *Solution Explorer*. Para que quede todo completo se tendrá que añadir la línea marcada en azul en la parte superior del código del nuevo programa tal y como se muestra en la figura 19.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using FinLib;
```

Figura 18

- Con esto ya si estaremos en disposición de usar todos los métodos de nuestra librería creada. Para poder usar los cuatro métodos únicamente se tendrá que hacer:

    - `FinLib.MarketRiskMetricsDT.getMarketRisk (...)`
    - `FinLib.MarketRiskMetricsDT.getMRContributions (...)`
    - `FinLib.MarketRiskMetricsDT.beta (...)`
    - `FinLib.MarketRiskMetricsDT.getActiveVector (...)`

Como se puede ver es sencillo utilizar los métodos.

6.6. Inclusión en el modelo PAT One

La introducción de la librería en el entorno *PAT One* ha sido realizada por una persona externa a la que ha realizado el proyecto. Se ha hecho de la siguiente manera[20]:

Se creó un controlador llamado *Market Risk Controller*. Éste se encarga de recibir los datos tanto de la base de datos como de los reportes *(GUI)*. Una vez se han recibido, el propio controlador se encargará de transformarlos de tal manera que permita realizar cada uno de los casos de uso para que puedan ser enviados al *Market Risk Engine*, lugar donde se encuentra la librería hecha y éste calcule los valores de riesgo que se deseen.

Una vez realizado los cálculos, el controlador *Market Risk Controller* se encargará también tanto de volcarlos en la base de datos como de reflejarlos en el reporte.

[20] Se recomienda seguir la figura 12 página 80 para facilitar el seguimiento

7. Conclusión

7.1. A nivel de aplicación

Una vez se ha finalizado el proyecto se puede hacer una valoración en frio tanto del proceso como del desarrollo del software final.

Para hacer una valoración del proceso basta con mirar los objetivos planteados. Se observa que se han cumplido todos los objetivos de cálculo marcados. Además, si se visualizan los requisitos no funcionales se aprecia que dentro de todas las librerías analizadas, el tiempo de respuesta de la ganadora es la menor de todas, así como los recursos y memoria utilizada.

También se han respetado los dos últimos apartados de los requisitos no funcionales donde no tenían que haber llamadas a la base de datos ni a ficheros ni a formatos específicos y el lenguaje de programación tenía que ser el estándar de la Compañía (C#)

Así pues se puede concluir que a nivel de la aplicación ha sido un éxito.

7.2. A nivel personal

La valoración final a nivel personal es sin duda positiva. Nuestra facultad se centra mucho en la informática descuidando quizás algunos aspectos básicos de otros campos. Así pues cuando me plantearon este proyecto me pareció interesante ya que aparte de aprender un nuevo lenguaje de programación, me ha aportado una serie de conocimientos económicos que si no hubiera emprendido este proyecto, jamás hubiera adquirido.

Por tanto una vez concluido el proyecto, puedo afirmar que la elección ha sido acertada ya que me he dotado de nuevos conocimientos, tanto informáticos como económicos que sin duda, serán de provecho en un futuro.

7.3. Problemas encontrados

Los problemas que han ido apareciendo en el proyecto no han sido de mucha importancia. La mayoría de ellos se deben al uso de una nueva librería para el testeo. No todas las librerías son iguales, eso implica que los métodos y los tipos de objetos para cada una son diferentes.

Se encontraba el caso de librerías que disponían de mucha documentación en la red no obstante otras era prácticamente nula la documentación, haciendo difícil algunos de los cálculos que se pretendían.

Hubo un problema con una de las librerías a la hora de hacer el transpuesto del vector. La librería en cuestión obtenía los valores de manera distinta a las otras que se analizaron y eso causo mucha confusión.

El último problema que apareció fue a la hora de hacer las pruebas con la librería final. Algunos de los cálculos obtenidos con la librería no eran los mismos que los que la Compañía disponía. Así pues, fue preciso hacer muchas pruebas para finalmente, darse cuenta que algunas de las fórmulas que se tenían inicialmente eran erróneas.

7.4. Futuras mejoras

El proyecto es moldeable permitiendo así futuras mejoras. Entre ellas se destacan las siguientes:

- Inclusión de nuevos parámetros de cálculo del riesgo de mercado.
- Posible cambio de librería. Si en un futuro se encontrará una librería mejor que *DotNumerics* sería fácil la modificación del proyecto para así usar la nueva.
- Es posible que se encuentren mejoras de optimización del código como ya se han incluido algunas. Éstas son:
    - Se pasó el vector resultante de la llamada getMarketRisk a la segunda llamada getMRContributions para que de esta manera se evitará la repetición de cálculos hechos.
    - Valores predefinidos para los niveles de confianza 0.95 y 0.99. Como son los más usados, la librería no tenía necesidad de calcularlos.
    - Almacenamiento de cálculos repetidos. Se daba el caso que algunos objetivos tenían en común algún parámetro de la operación dentro del mismo método. Al guardarlos en una variable, se calculaba una única vez ahorrando así tiempo de respuesta final.

8. Bibliografía

A continuación se puede ver la bibliografía separada en dos apartados en función de donde se haya obtenido. Puede ser por una fuente escrita o bien por una fuente virtual.

8.1. Libros y *papers*

- **C.Hull, John,** Options*, Futures, and Other Derivative, Enero 2000*
- **Fabozzi, Frank J.,** *the Handbook of Fixed Income Securities, Abril 2005*
- **Sharp, John and Jagger, Jon,** *Microsoft Visual C#. Net Step By Step, 2003.*
- **Del Canto, Ángel and Delfiner, Miguel,** *Minimum capital requirements for Market Risk, Enero 2009.*
- **Down, Kevin,** *Measuring Market Risk, 2005.*
- **García Estévez, Pablo,** *El valor en Riesgo (VaR). Instituto de Empresa. Fecha de publicación desconocida*

8.2. Páginas web

- **Metodología SCRUM:** Conjunto de páginas donde se explica esta metodología de seguimiento de proyectos:

  http://pymecrunch.com/scrum-metodologia-agil-para-tus-proyectos. Marzo 2011
  http://www.chuidiang.com/ood/metodologia/scrum.php. Marzo 2011

- **Librerías:** Página donde se exponen diferentes librerías candidatas a ser utilizadas:
  http://stackoverflow.com/questions/4159385/anyone-have-any-experience-with-dotnumerics-alglib-dnanalytics-math-net-f-f. Marzo 2011
- **DotNumerics:** Página oficial de la librería de nuestro proyecto:
  http://www.dotnumerics.com/. Marzo 2011
- **DotNumerics Clases:** Se muestran las clases que contiene la librería de nuestro proyecto:
  http://nd-voronoi-sharp.googlecode.com/svn-history/r8/trunk/dependencies/DotNumerics.XML. Marzo 2011

- **Diccionario financiero:** Páginas web sobre un diccionario financiero muy completo: *http://financial-dictionary.thefreedictionary.com/*. Marzo 2011
  http://www.economia48.com/. Marzo 2011

- **Benchmarking:** Definición de términos.
  http://www.degerencia.com/tema/benchmarking. Marzo 2011

- **Librerías Benchmarking:** Librería para el cálculo de consumo de CPU.
  http://casidiablo.net/capturar-informacion-sistema-operativo-java/. Marzo 2011

- **Iber Mutua:** Cálculo de impuestos para trabajadores autónomos.
  http://www.ibermutuamur.es/spip.php?page=formulario-cuota-autonomo. Enero 2011

- **Java SE:** Página oficial de Java Standard Edition de Oracle.
  http://www.oracle.com/technetwork/java/javase/downloads/index.html. Octubre 2011

- **Visual C#. net:** Página oficial de Microsoft Visual C#. Net
  http://msdn.microsoft.com/es-es/vcsharp/aa336706. Abril 2011

- **Tablas de Hash en C#:** Página donde se explica cómo tratar las tablas de dispersión en C#:
  http://developmania.wordpress.com/2008/09/14/colecciones-de-datos-en-cnet-parte-iv-hashtable/. Abril 2011

- **Netbeans IDE 6.9.1:** Página oficial del IDE Netbeans.
  http://netbeans.org/. Enero 2011

- **Market Risk Beta:** Definición del parámetro Beta del *Market Risk:*
  http://www.investing-in-mutual-funds.com/market-risk.html. Marzo 2011

- **Implied Alpha:** Definición de la *Implied Alpha:*
  http://zantrio.com/dictionary/implied_alpha.php. Marzo 2011

- **Google Académico:** Página que contienen muchísimos artículos académicos.
  http://scholar.google.es/. Febrero 2011

- **Matrices en Excel:** Página donde se ha encontrado una solución para hacer multiplicación dada una matriz, un vector y un vector transpuesto:
  http://es.answers.yahoo.com/question/index?qid=20070620140348AAf3zgo. Abril 2011

- **Catálogo biblioteca UPC:** Se ha recogido varios proyectos finales para ver la estructura así como los formatos para servir de modelo para este documento.
  http://cataleg.upc.edu/. Marzo 2011

9. Anexos

9.1. Tabla de procesos en ejecución

Image Name	User Name	CPU	Memory (Private Working Set)	Description
atieclxx.exe	SYSTEM	00	2,240 K	AMD External Events Client Module
atiesrxx.exe	SYSTEM	00	1,648 K	AMD External Events Service Module
ccApp.exe *32	Xguirao	00	456 K	Symantec User Session
ccSvcHst.exe *32	SYSTEM	00	1,112 K	Symantec Service Framework
csrss.exe	SYSTEM	00	3,176 K	Client Server Runtime Process
csrss.exe	SYSTEM	00	6,972 K	Client Server Runtime Process
dwm.exe	Xguirao	00	16,860 K	Desktop Window Manager
explorer.exe	Xguirao	00	32,572 K	Windows Explorer
jucheck.exe *32	Xguirao	00	2,184 K	Java(TM) Update Checker
jusched.exe *32	Xguirao	00	1,488 K	Java(TM) Update Scheduler
lsass.exe	SYSTEM	00	6,536 K	Local Security Authority Process
lsm.exe	SYSTEM	00	2,688 K	Local Session Manager Service
MonitoringHost.exe	SYSTEM	00	6,864 K	System Center Management Service Host Pro...
MonitoringHost.exe	SYSTEM	00	3,012 K	System Center Management Service Host Pro...
PresentationFontCache.exe	LOCAL ...	00	6,028 K	PresentationFontCache.exe
ProtectionUtilSurrogate.exe *32	Xguirao	00	3,560 K	Symantec AntiVirus
ReportingServicesService.exe	NETWO...	00	47,304 K	Reporting Services Service
Rtvscan.exe *32	SYSTEM	00	2,008 K	Symantec AntiVirus
SearchIndexer.exe	SYSTEM	00	48,704 K	Microsoft Windows Search Indexer
services.exe	SYSTEM	00	6,980 K	Services and Controller app
Smc.exe	SYSTEM	00	4,072 K	Symantec CMC Smc
SmcGui.exe	Xguirao	00	2,264 K	Symantec CMC SmcGui
smss.exe	SYSTEM	00	496 K	Windows Session Manager
spoolsv.exe	SYSTEM	00	9,604 K	Spooler SubSystem App
svchost.exe	LOCAL ...	00	7,708 K	Host Process for Windows Services
svchost.exe	SYSTEM	00	4,596 K	Host Process for Windows Services
svchost.exe	NETWO...	00	6,188 K	Host Process for Windows Services
svchost.exe	SYSTEM	00	5,564 K	Host Process for Windows Services
svchost.exe	LOCAL ...	00	13,224 K	Host Process for Windows Services
svchost.exe	SYSTEM	00	166,000 K	Host Process for Windows Services
svchost.exe	SYSTEM	00	36,712 K	Host Process for Windows Services
svchost.exe	NETWO...	00	9,536 K	Host Process for Windows Services
svchost.exe	LOCAL ...	00	6,968 K	Host Process for Windows Services
svchost.exe	SYSTEM	00	4,408 K	Host Process for Windows Services
svchost.exe	LOCAL ...	00	4,540 K	Host Process for Windows Services
svchost.exe	SYSTEM	00	1,504 K	Host Process for Windows Services
svchost.exe	NETWO...	00	2,052 K	Host Process for Windows Services
System	SYSTEM	00	668 K	NT Kernel & System
System Idle Process	SYSTEM	99	24 K	Percentage of time the processor is idle
taskeng.exe	Xguirao	00	1,996 K	Task Scheduler Engine
taskhost.exe	Xguirao	00	2,996 K	Host Process for Windows Tasks
taskmgr.exe	Xguirao	00	3,652 K	Windows Task Manager
vmware-usbarbitrator.exe *32	SYSTEM	00	1,268 K	VMware USB Arbitration Service
wininit.exe	SYSTEM	00	1,412 K	Windows Start-Up Application
winlogon.exe	SYSTEM	00	2,636 K	Windows Logon Application

Figura 19

9.2. Tabla de consumo CPU

Tabla donde se produce un ejemplo de salida donde mediante la clase de java InfoCPU se obtenían todas las medidas que necesitábamos para el test de consumo de nuestra CPU.

CPU 0	CPU 1	CPU 2	CPU 3	Consumo Total CPU	Media
0.00%	3.20%	0.00%	0.00%	0.70%	0.007
0.00%	0.00%	0.00%	3.70%	1.00%	0.01
0.00%	0.00%	0.00%	0.00%	0.00%	
6.20%	4.10%	0.00%	2.10%	3.10%	0.031
0.00%	0.00%	0.00%	0.00%	0.00%	
0.00%	0.00%	0.00%	2.00%	0.40%	0.004
0.00%	0.00%	0.00%	0.00%	0.00%	
0.00%	0.00%	0.00%	2.10%	0.50%	0.005
0.00%	0.00%	0.00%	2.10%	0.50%	0.005
0.00%	0.00%	0.00%	2.00%	0.40%	0.004
0.00%	0.00%	0.00%	0.00%	0.00%	
0.00%	0.00%	0.00%	0.00%	0.00%	
0.00%	0.00%	0.00%	0.00%	0.00%	
0.00%	0.00%	0.00%	2.00%	0.50%	0.005
0.00%	0.00%	0.00%	0.00%	0.00%	
0.00%	2.10%	0.00%	0.00%	0.50%	0.005
2.10%	0.00%	2.00%	0.00%	1.00%	0.01
0.00%	0.00%	0.00%	0.00%	0.00%	
0.00%	0.00%	0.00%	0.00%	0.00%	
0.00%	0.00%	0.00%	0.00%	0.00%	
			Máximo	3.10%	0.86%

Tabla 11

9.3. Tabla de consumo de memoria RAM

La siguiente tabla muestra un ejemplo del *output* de datos obtenidos de la clase java InfoMemoria. Se obtendrán los datos tanto de la memoria RAM como de la memoria SWAP.

RAM	RAM Delta	RAM Total	RAM Used	RAM available	SWAP total	SWAP Used	SWAP available
4096	28	4193268	1920604	2272664	10479808	2501532	7978276
4096	-36	4193268	1920576	2272692	10479808	2501572	7978236
4096	-72	4193268	1920504	2272764	10479808	2502432	7977376
4096	4	4193268	1920508	2272760	10479808	2502432	7977376
4096	8288	4193268	1928796	2264472	10479808	2528100	7951708
4096	-164	4193268	1928632	2264636	10479808	2528168	7951640
4096	20	4193268	1928652	2264616	10479808	2528356	7951452
4096	72	4193268	1928724	2264544	10479808	2528592	7951216
4096	0	4193268	1928724	2264544	10479808	2528672	7951136
4096	116	4193268	1928840	2264428	10479808	2528708	7951100
4096	-4	4193268	1928836	2264432	10479808	2528756	7951052
4096	-40	4193268	1928796	2264472	10479808	2528956	7950852
4096	36	4193268	1928832	2264436	10479808	2528796	7951012
4096	-12	4193268	1928820	2264448	10479808	2528652	7951156
4096	-4	4193268	1928816	2264452	10479808	2528572	7951236
4096	-396	4193268	1928420	2264848	10479808	2528480	7951328
4096	0	4193268	1928420	2264848	10479808	2528480	7951328
4096	-6644	4193268	1921776	2271492	10479808	2503192	7976616
4096	-356	4193268	1921420	2271848	10479808	2503088	7976720
4096	0	4193268	1921420	2271848	10479808	2503088	7976720

Tabla 12

9.4. Herramientas extras para realizar el proyecto

A continuación se exponen herramientas secundarias que se han usado en el proyecto. No se han introducido dentro de la memoria porqué se considera que sin su uso, el proyecto funcionaria igualmente.

- **Tortoise***:* Aplicación utilizada para hacer un control de las diferentes versiones que va teniendo el software, desde la creación hasta el final. Se ha utilizado en este proyecto para alojar todo el proyecto en un repositorio propio de la empresa y que se pueda trabajar con él desde cualquier oficina.
  http://tortoisesvn.tigris.org/

- **Google Chrome:** Navegador web utilizado.
  http://www.google.com/chrome?hl=es

- **Microsoft Outlook 2010:** Herramienta que permite el intercambio de mails, en particular con el director de proyecto, para poder realizar un seguimiento del proceso y un intercambio de información. Se incluye en el paquete de Microsoft Office 2010.

- **Microsoft Word 2010:** Herramienta ofimática por excelencia y donde se ha escrito la documentación del proyecto.

- **Microsoft Excel 2010:** Herramienta ofimática donde se ha hecho el volcado de los datos obtenidos en el benchmarking y donde se han realizado los gráficos para hacer la comparativa.

- **Visual Studio 2010 Feature Pack:** Pack donde se dispone de la herramienta para crear el diagrama de clases de cada uno de los proyectos realizados en el Microsoft Visual C#.Net.