UNIVERSIDAD DE CHILE

FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS

DEPARTAMENTO DE INGENIERÍA QUÍMICA Y BIOTECNOLOGÍA

DISEÑO E IMPLEMENTACIÓN DE UN ALGORITMO INTEGRATIVO PARA GUIAR LA MUTACION DE SECUENCIAS CODIFICANTES DE ADN PARA MAXIMIZAR SU EXPRESIÓN

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL QUÍMICO

JAIME ANDRÉS URZÚA GÓMEZ

PROFESOR GUÍA:
ÁLVARO OLIVERA NAPPA

MIEMBROS DE LA COMISIÓN:
M. ORIANA SALAZAR AGUIRRE
BARBARA A. ANDREWS

SANTIAGO DE CHILE
2014

RESUMEN DE LA MEMORIA PARA OPTAR AL
TÍTULO DE: Ingeniero Civil Químico
POR: Jaime Andrés Urzúa Gómez
FECHA: 12/09/2014
PROFESOR GUÍA: Álvaro Olivera Nappa

DISEÑO E IMPLEMENTACIÓN DE UN ALGORITMO INTEGRATIVO PARA GUIAR LA MUTACION DE SECUENCIAS CODIFICANTES DE ADN PARA MAXIMIZAR SU EXPRESIÓN

El presente trabajo tiene como objetivo principal, idear e implementar un algoritmo para guiar el diseño de proteínas recombinantes. Para lo anterior se recopiló y conformó una base de datos de uso de codones y abundancia de ARNt para un variado grupo de organismos, que incluyó eucariontes y procariontes. El algoritmo opera sobre una secuencia de ADN codificante, de manera que: identifica, analiza cuantitativa y cualitativamente, sugiere mutaciones de acuerdo a las bases de datos designadas automática o manualmente, analiza el plegamiento del ARNm, posibilita la disminución de la estabilidad de estructuras secundarias y permite la visualización gráfica del plegamiento. El algoritmo de mutación trabaja bajo el criterio que el sesgo en el uso de codones refleja la selección para la optimización del proceso de traducción guiado por la abundancia de ARNt. El algoritmo está diseñado para maximizar la expresión recombinante de la proteína codificada sin cambiar su secuencia de aminoácidos. Las mutaciones sugeridas son específicas para cada gen y para cada organismo hospedero en donde dicho gen se desea expresar.

La principal tarea y mayor desafío del presente trabajo fue trascender los mecanismos de evolución y los inconvenientes asociados a la producción de proteínas recombinantes. Esto se realizó mediante la información y evaluación de resultados experimentales de diversas investigaciones focalizadas a la genética molecular. De este modo, se logró complementar estudios desarrollados en: el uso de codones, abundancia de ARNt, baja eficiencia de traducción en los primeros ~30-50 codones del ARNm, apareamiento de codones sinónimos en el ARNm, uso de codones mayoritarios y formación de estructuras secundarias en el ARNm. Estas características de presencia transversal en la biología celular, se utilizaron como base en el diseño e implementación de la presente herramienta bioinformática, para el análisis de secuencias codificantes de ADN y mutación sinónima en el proceso de optimización de la expresión de proteínas recombinantes.

Tabla de contenido

Índice de tablas

Índice de figuras

Nomenclatura y Abreviaturas

ADN:	Ácido desoxirribonucleico: macromolécula que codifica los genes de un organismo.
ARNm:	Ácido ribonucleico mensajero: moléculas lineales encargadas de transferir la información genética del ADN
ARNr:	Ácido ribonucleico ribosomal: moléculas encargadas de la función catalítica de los ribosomas.
ARNt:	Ácido ribonucleico de transferencia: moléculas que hacen posible la traducción de una secuencia de ARNm a una secuencia de aminoácidos.
ATP:	Adenosine Triphosphate. Adenosín trifosfato: molécula utilizada como fuente de energía.
Codón:	Triplete de nucleótidos.
CTCPB:	co-tRNA Codon Pairing Bias. Sesgo en el uso pareado de codones por la utilización de un mismo ARNt.
CUB:	Codon Usage Bias. Sesgo en el uso de codones.
DSAM:	DNA Sequence Analysis and Mutations. Análisis y mutación de secuencias de ADN.
GC%:	Porcentaje de Guanina y Citosina en una determinada sección del ADN.
GUI:	Graphical User Interface. Interfaz gráfica de usuario: herramienta de MATLAB para la confección de interfaces.
H_0:	Hipótesis: Codones en la secuencia del gen mutados por la herramienta bioinformática de análisis de secuencias de ADN.
H_1:	Hipótesis: Codones en la secuencia del gen no mutados por la herramienta bioinformática de análisis de secuencias de ADN.
MATLAB:	Matrix Laboratory. Laboratorio de matrices: software de programación utilizado en el presente trabajo.
MCU:	Major Codon Usage. Uso de codones mayoritarios.
ORF:	Open Reading Frame. Marco abierto de lectura: sección de ADN o ARN que contiene la información necesaria para la expresión de una proteína.
PA:	Protein Abundance. Abundancia de proteína.
Sitio E:	Sitio en la subunidad mayor del ribosoma, donde se libera el ARNt.
UTR:	Untraslated Region. Región sin traducir: sectores colindantes a los ORF, que no son traducidos.
$\Delta G°$:	Diferencia de energía libre de Gibbs.

I Introducción, Antecedentes y Objetivos

1.1 Introducción

La expresión de genes es uno de los principales procesos moleculares en las células vivas. Los organismos invierten una considerable cantidad de recursos, incluyendo energía, materia prima y capacidad de manejar un flujo de información, al mismo tiempo que optimizan la eficiencia y precisión del proceso. La evolución ha permitido que los organismos desarrollen sofisticados métodos para alcanzar esas metas, y para balancear estos objetivos cuando es necesario, por ejemplo cuando los recursos se vuelven escasos.

La producción e ingeniería de proteínas recombinantes procura que el proceso de expresión se realice con una alta eficiencia y fidelidad, y de este modo, contribuir y preservar un correcto funcionamiento de las células.

La redundancia del código genético, por cual la mayoría de los aminoácidos pueden ser traducidos a partir de más de un codón distinto, abre la oportunidad de optimizar la eficiencia y precisión de la producción de la proteína en varios niveles, manteniendo la misma secuencia de aminoácidos pero cambiando los codones que codifican por ellos (Gingold y Pilpel, 2011). Sin embargo, esta optimización debe tener en cuenta que el uso de codones sinónimos para un mismo aminoácido, que a menudo es correctamente subentendido como un cambio neutral o silencioso, ocurre en la naturaleza de modo no aleatorio y se ha demostrado en un creciente número de casos, que es el resultado de una selección natural, que actúa, principalmente, para mejorar la eficiencia y precisión de la traducción (Drummond y Wilke, 2008; Cannarozzi et al, 2010; Tuller et al, 2010).

A lo anterior se suma la disminución considerable de los costos de síntesis de secuencias de ADN artificiales, alcanzando a ser incluso en determinados casos, más convenientes económicamente que los métodos clásicos de ingeniería genética (Raab et al, 2009). Esto establece un escenario ideal para el diseño, ejecución de pruebas empíricas y posterior producción de proteínas recombinantes con secuencias codificantes modificadas que garantice su producción con alta eficiencia y fidelidad.

Ante tal escenario, surge la necesidad de una herramienta que, conjugando los más recientes descubrimientos en genética molecular, examine y procese secuencias de genes, y permita sugerir mutaciones silentes o genes artificiales que codifiquen la misma proteína, cambiando codones de una secuencia original por codones sinónimos, de manera de aumentar la eficiencia de la producción de proteínas recombinantes en diversos organismos.

1.1.1 Descripción General y Conceptos Básicos

El proceso de expresión de genes, a grandes rasgos, se inicia con la transcripción de una hebra molde de ADN a ARN mensajero (ARNm) (ver Figura 1). Posteriormente este ARNm es traducido en los ribosomas, usando moléculas de ARN de transferencia (ARNt) específicas, para formar la cadena de aminoácidos correspondientes y así expresar el gen (ver Figura 2).

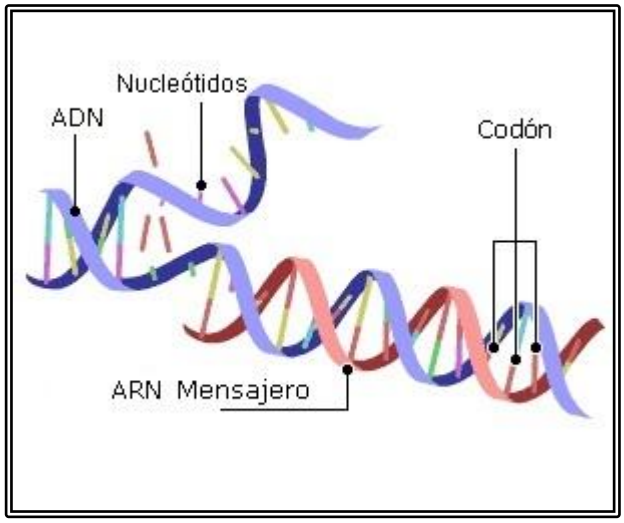

Figura 1: Esquema de ADN y ARN

Los genes se encuentran codificados en segmentos del ADN, en la forma de secuencias de nucleótidos, los que organizados en tripletes codifican una determinada secuencia de aminoácidos. Los tripletes de nucleótidos son conocidos como codones. El ADN está constituido por 4 nucleótidos. La combinación de éstos puede dar origen a un total de 64 codones diferentes. Los nucleótidos en el ADN son: adenina (A), guanina (G), citosina (C) y timina (T). Por su parte los nucleótidos en el ARN son, respectivamente: adenina (A), guanina (G), citosina (C) y uracilo (U).

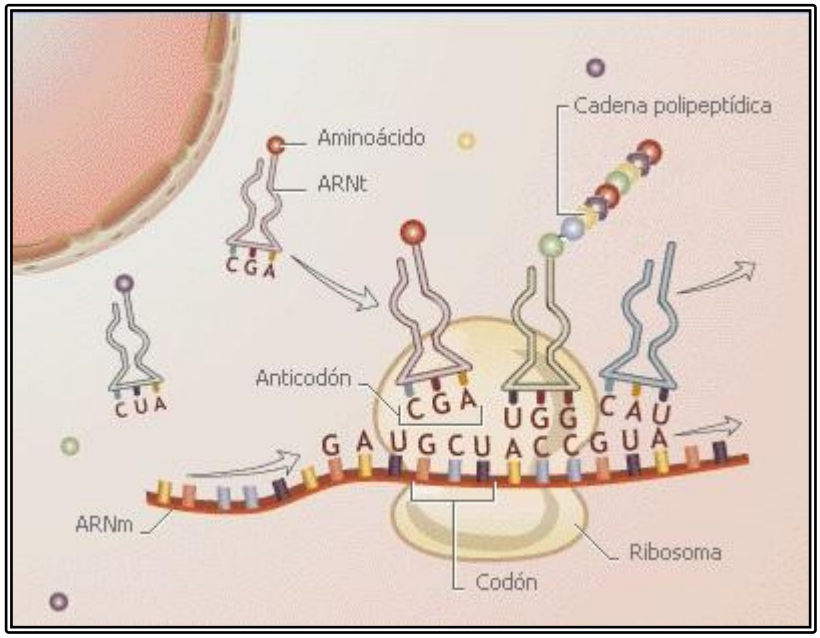

Figura 2: Esquema de ARNm, ARNt y Ribosoma

Los 64 codones posibles son traducidos de la misma manera en todos los seres vivientes, con sólo algunas excepciones. De los 64 codones posibles de este código genético universal, sólo 61 codifican aminoácidos (ver Tabla 1). Los 3 restantes son codones de terminación de la traducción, conocidos como codones de término (TAA,

TAG y TGA). Por su parte, el codón de inicio de la traducción es ATG y codifica el aminoácido, metionina. A excepción de la metionina y el triptófano que están codificados por un único codón, los aminoácidos pueden estar codificados por 2, 3, 4 ó 6 codones diferentes. Los codones que codifican un mismo aminoácido son, usualmente, considerados “sinónimos”.

		Segundo Nucleótido									
Primer Nucleótido		T		C		A		G			Tercer Nucleótido
	T	TTT	Fenilalanina	TCT	Serina	TAT	Tirosina	TGT	Cisteína	T	
		TTC		TCC		TAC		TGC		C	
		TTA	Leucina	TCA		TAA	Codón Parada	TGA	C. Parada	A	
		TTG		TCG		TAG		TGG	Triptófano	G	
	C	CTT	Leucina	CCT	Prolina	CAT	Histidina	CGT	Arginina	T	
		CTC		CCC		CAC		CGC		C	
		CTA		CCA		CAA	Glutamina	CGA		A	
		CTG		CCG		CAG		CGG		G	
	A	ATT	Isoleucina	ACT	Treonina	AAT	Asparagina	AGT	Serina	T	
		ATC		ACC		AAC		AGC		C	
		ATA		ACA		AAA	Lisina	AGA	Arginina	A	
		ATG	Metionina	ACG		AAG		AGG		G	
	G	GTT	Valina	GCT	Alanina	GAT	Acido Aspártico	GGT	Glicina	T	
		GTC		GCC		GAC		GGC		C	
		GTA		GCA		GAA	Acido Glutámico	GGA		A	
		GTG		GCA		GAG		GGG		G	

Tabla 1: Codones y aminoácidos codificados en el código genético universal

De un total de 20 aminoácidos existentes, 18 son codificados por 2 o más codones sinónimos. Sin embargo, usualmente, los codones sinónimos se utilizan de modo dispar en el genoma. Tal sesgo en el uso de codones se denomina *Codon Usage Bias,* CUB (Hershberg y Petrov, 2009; Ikemura, 1985). Es así como en los genomas, los genes altamente expresados tienden a tener un alto CUB en comparación con los débilmente expresados (Ikemura, 1981). Este incremento del sesgo en el uso de codones relacionado con los niveles de expresión de los genes ha sido observado en organismos de todos los dominios de la vida.

1.1.2 ARN mensajero

La traducción del ARNm consta de cuatro fases: iniciación, elongación, terminación y separación del ribosoma. En la mayoría de los casos la iniciación de la traducción es la etapa limitante de la velocidad de traducción (Salis et al, 2009).

El control de la traducción es realizado en múltiples etapas y por diversos mecanismos. Sin embargo, la mayor parte del control ocurre en la etapa de iniciación, donde los ribosomas se unen al ARNm, típicamente en la región no traducida 5’ (UTR) (Ingolia et al, 2009). La fase de elongación es gobernada por la estructura secundaria del ARNm (Gray y Hentze, 1994) y el grado de adaptación de la secuencia codificada a la abundancia de ARNt celular (dos Reis et al., 2004; Sharp y Li, 1987).

Por otro lado, los costos del proceso de traducción son numerosos tanto en energía como en la asignación de los recursos celulares, tales como los ribosomas y los ARNt (Stoebel et al, 2008).

1.1.3 ARN de transferencia

La abundancia de ARNt es un factor importante en el comportamiento genético molecular durante la traducción. La abundancia de los ARNt correspondientes a los diferentes codones en un gen puede determinar la velocidad (Akashi, 2003; Man y Pilpel, 2007) y la precisión de la traducción (Drummond y Wilke, 2008). Los transcriptos cuyos codones están sesgados hacia los ARNt más abundantes son expresados en mayor cantidad (Man y Pilpel, 2007; Qin et al, 2004). En este mismo sentido, los codones sinónimos y sus correspondientes ARNt pueden diferir en cuanto a las cantidades presentes en las células, y esto también influye en las velocidades a las que serán reconocidos por el ribosoma (Varenne et al, 1984; Sorensen et al, 1989). Un buen predictor de la abundancia de cada ARNt particular es el número de copias de genes de dicho ARNt (dos Reis et al., 2004)

Otros estudios también sugieren que la traducción, y así también la abundancia de proteína (PA), están correlacionadas con el nivel de adaptación a la abundancia de ARN de transferencia (Tuller et al, 2010), con un débil plegamiento del ARNm, con el principio del marco abierto de lectura (ORF), con el largo del ORF, con el contenido de guanina-citosina (GC) y con varias características secundarias del extremo 5’ de la región sin traducir (UTR) del gen (Zur y Tuller, 2012).

1.2 Antecedentes

La construcción e implementación de una herramienta bioinformática de análisis de secuencias de ADN para sugerir mutaciones silentes, enfrenta una amplia gama de posibilidades, así como de dificultades, desde su concepción hasta su finalización.

Dentro de las posibilidades está el hecho que una secuencia de aminoácidos de una proteína, presenta múltiples grados de libertad que pueden permitir cambios en dicha secuencia para mejorar la eficiencia de traducción de cada gen bajo variadas condiciones y tipos de células. Precisamente es la redundancia del código genético, la que permite la selección de codones alternativos para el mismo aminoácido.

Las dificultades que enfrenta dicha herramienta se concentran en las mutaciones en sí mismas, que a pesar de ser “sinónimas”, pueden ejercer efectos drásticamente distintos en el proceso de traducción (Gingold y Pilpel, 2011). Se ha reportado que el reemplazo de codones sinónimos puede cambiar la estructura de la proteína y su función, lo que indica que la estructura de la proteína depende de la secuencia especifica de ADN que la codifica (Angov et al, 2008). También se ha descrito que las sustituciones de codones sinónimos que cambian la frecuencia del uso de codones de infrecuentes a frecuentes en regiones de lenta traducción del ARNm puede tener efectos perjudiciales en la actividad enzimática (Komar, 1999).

Así, contrariamente a lo que se piensa, convencionalmente, las sustituciones de codones sinónimos pueden no ser siempre silentes. El cambio de la frecuencia del uso de codones puede afectar la estructura y función de la proteína. Además, la frecuencia de uso de codones otorga vital información sobre la formación de estructuras secundarias y terciarias de la proteína (Angov et al, 2008).

La importancia de la evolución natural y la identificación de los patrones naturales de optimización, toma mayor relevancia a la hora de diseñar una herramienta de análisis y optimización de secuencias de ADN. Los factores a considerar deben apuntar a la precisión, que puede ser descrita como la probabilidad de que la proteína traducida esté libre de errores y coincida con la secuencia prescrita por la secuencia del código genético, en adición con la probabilidad de que se pliegue correctamente en la célula (Drummond y Wilke, 2008; Zhou et al, 2009). Además, dicha optimización natural está orientada a obtener un nivel óptimo de expresión de proteína, donde el beneficio de la expresión del gen excede el costo de su producción (Dekel y Alon, 2005).

Los patrones en la composición de bases en el ADN y la variación en las secuencias de ADN, sugieren que la selección natural discrimina entre los codones sinónimos para mejorar la síntesis de proteína (Andersson y Kurland, 1990; Sharp et al, 1993). Adicionalmente, las posiciones de los nucleótidos sinónimos son esenciales para el mantenimiento y función de diversas señales reguladoras localizadas en las regiones codificantes de proteínas (Shabalina et al, 2012).

Teniendo en consideración lo anterior, el diseño de una herramienta bioinformática que analice un marco abierto de lectura y sugiera mutaciones sinónimas para optimizar la producción de la proteína recombinante codificada, debe incluir varios factores que influyen en la expresión óptima. Para los objetivos de este trabajo, se ha seleccionado los factores analizados en las siguientes subsecciones.

1.2.1 Rampa inicial de baja eficiencia de traducción

Un perfil de eficiencia de traducción en un gen está definido para cada codón, como la disponibilidad estimada de los ARNt que participan en la traducción de cada uno (Tuller et al, 2010).

Un estudio relacionado a lo anterior descubrió un perfil de eficiencia universal de la traducción que se caracteriza por tener una baja eficiencia de traducción en los primeros ~30-50 codones de los ARNm. Esta característica se conserva en las especies que representan los tres dominios de la vida. Este perfil universal define que la generalidad de las transcripciones comienza con codones con una eficiencia relativamente baja, por alrededor de las primeras 30 a 50 posiciones. Esta zona se definió como “rampa de baja eficiencia” o simplemente “rampa” (Tuller et al, 2010).

La rampa limita la velocidad de los ribosomas sobre las primeras decenas de codones en las trascripciones, y genera como consecuencia una zona de alta densidad ribosomal. La generación de una sección corta de alta densidad en la región 5’ de los ARNm puede dar lugar a una región libre de atascos en el resto de la transcripto, dado que los ribosomas que pasan este cuello de botella tienen menor probabilidad de atascarse.

De esta forma, esta rampa tiene el potencial de reducir los embotellamientos de los ribosomas en la traducción del resto del gen. La reducción del número de atascos es deseable por varias razones: primero, reduce la cantidad de tiempo total en que los

ribosomas son retenidos por una transcripto y, en segundo lugar, los ribosomas atascados que se detienen en codones lentos, emplean más tiempo en la transcripción, incrementando la probabilidad de un desprendimiento espontáneo (Li et al., 2006).

La propiedad de mayor importancia de una rampa, es que puede hacer que la mayoría de los abortos en la traducción ocurran al principio de los transcriptos (Tuller et al, 2010). Esto es deseable porque en esas regiones un desprendimiento espontáneo es menos costoso en términos de energía (ATP) y materias primas (p.ej., ARNt cargados). Por lo tanto la importancia de la existencia de las rampas radica en la disminución de la probabilidad de atasco, con la consecuente reducción del costo de la expresión del gen e incremento de la capacidad de producción a ese costo.

Es importante señalar que la longitud promedio de la rampa en eucariontes es 34.5 codones, mientras que en procariontes es 24 codones. Esto puede corresponder a las diferencias en el tamaño de la región englobada por los ribosomas procarióticos y eucarióticos (Tuller et al, 2010).

1.2.2 Bases de datos de uso de codones y abundancia de ARNt

Parte fundamental del diseño de una herramienta de optimización de genes heterólogos son las bases de datos. La base de datos de uso de codones se obtuvo del estudio de las frecuencias de 257.468 secuencias codificadoras de proteínas completas, que fueron recopiladas de las divisiones taxonómicas de la base de datos de secuencias de ADN de GenBank. Con lo anterior se calculó la suma de los codones utilizados por 8792 organismos (Nakamura et al, 1999).

Además, se utilizó una base de datos de abundancia de ARNt, que se construyó mediante la identificación y cuantificación de los genes de ARNt (Lowe y Eddy, 1997).

Estas bases de datos se emplearon como insumo de las herramientas predictivas desarrolladas en este trabajo.

1.2.3 Uso pareado de codones en el ARNm

Se ha descrito la existencia de un fenómeno de sesgo en el uso pareado de codones por la utilización de un mismo ARNt (*co-tRNA Codon Pairing Bias*, CTCPB). En efecto, se ha observado que cuando un codón particular es traducido, la subsecuente presencia del mismo aminoácido en la secuencia codificada tiende a utilizar el mismo ARNt (Shao et al, 2012), lo que produce un sesgo de uso de codones específico para cada gen.

En una investigación para la identificación de las preferencias de pareamiento entre los codones sinónimos sobre un total de 773 genomas bacterianos, se detectó un sesgo de pareamiento evidente hacia los pares de codones idénticos. Esto es consistente con el fenómeno de CTCPB, dado que los codones idénticos son, ciertamente, leídos por el mismo ARNt. Se detectó un alto nivel de pareamiento de codones sinónimos en el 73% de las especies bacterianas investigadas, lo que sugiere la existencia de una estrategia

de ordenamiento de codones sinónimos para mejorar la eficiencia en la traducción de procariontes (Shao et al, 2012).

Lo anterior, sumado a la presencia del fenómeno en secuencias codificantes de levaduras y de algunos otros eucariontes, posibilita adoptar esta estrategia como método para mejorar la eficiencia de traducción de proteínas recombinantes. Esta elección de codón es beneficiosa porque el ARNt no difunde lejos del ribosoma después de salir de su sitio E y es reutilizado para la traducción del próximo codón sinónimo cuando el complejo terciario se forma nuevamente (Cannarozzi et al, 2010).

1.2.4 Uso de codones mayoritarios (MCU)

Entre los codones sinónimos reconocidos por múltiples ARNt, los codones mayoritarios tienden a ser decodificados por el ARNt más abundante para un determinado aminoácido. Entre los codones reconocidos por ARNt con mayor abundancia, el codón que forma un emparejamiento del tipo Watson-Crick con el anticodón del ARNt es favorecido (Ikemura, 1982; Bennetzen y Hall, 1982) y catalogado como un codón mayoritario.

Aunque los patrones generales del uso de codones se correlacionan con la abundancia de ARNt en muchas especies, el uso de codones varía considerablemente entre los genes dentro de sus genomas. Sin embargo, el uso de codones mayoritarios muestra una fuerte correlación con los niveles de expresión génica en *Escherichia coli*, levaduras, hongos, *Drosophila melanogaster* y en genomas de cloroplastos de numerosas plantas, entre otros. (Akashi y Eyre-Walker, 1998). Es así, como en *Escherichia coli* y *Saccharomyces cerevisiae*, la composición de las bases de todo el genoma en sitios silentes, está sesgada hacia el subconjunto de los codones “mayoritarios” para cada aminoácido. (Akashi y Eyre-Walker, 1998).

De lo anterior se desprende que el uso de codones mayoritarios puede mejorar la adaptabilidad y capacidad de replicación, al aumentar las tasas de crecimiento y/o reducir los costes metabólicos de la síntesis de la proteína (Akashi y Eyre-Walker, 1998).

1.2.5 Plegamiento del ARNm

El ARNm hasta hace unos años era visto, únicamente, como un portador del código genético, transmitiendo la información de la secuencia primaria de aminoácidos entre los genes codificantes en el ADN y las proteínas. Sin embargo, recientes estudios revelan un rol significativo del ARNm en la regulación de la complejidad biológica. Esta facultad atribuida al ARNm deriva directamente de la redundancia del código genético.

Numerosas investigaciones señalan a los sitios sinónimos como responsables del rol regulador del ARNm. Al estudiar su impacto en la funcionalidad de los genes, se encontró efectos: en el corte y empalme del ARNm, en el plegamiento del ARNm, y en la estabilidad y la regulación de la traducción, a través de la utilización de los codones sinónimos “preferidos” que se traducen de manera más eficiente y exacta (Shabalina et al, 2012).

La estructura secundaria del ARN está compuesta principalmente por regiones de ARN de doble cadena originadas por plegamiento de la molécula lineal sobre sí misma (ver Figura 3). Al combinarse regiones de hebra simple y doble, la energía liberada por el apareamiento de bases incrementa la estabilidad energética de la molécula, mientras que las bases libres la desestabilizan.

Lo anterior hace del plegamiento del ARNm, un punto crucial a tener en cuenta al realizar mutaciones en un gen heterólogo. Esto se debe a que el cambio de bases puede dar lugar a prolongadas regiones de doble cadena, lo que aumentaría la estabilidad energética de la molécula de ARNm. La conservación estable de estos elementos plegados, puede afectar a la traducción y, en última instancia, la estructura y función de las proteínas (Shabalina et al, 2006; Parmley y Hurst, 2007).

Para evitar problemas en la traducción o en pasos posteriores, se sugiere un patrón periódico en la estructura secundaria del ARNm, que establezca una estructura de las transcriptos más ordenada y estable en las regiones codificantes de proteínas. La importancia funcional de las posiciones sinónimas para el mantenimiento de estructuras de ARN estables, son cruciales para la regulación de la expresión de proteínas, especialmente en la iniciación de la traducción. (Shabalina et al, 2006).

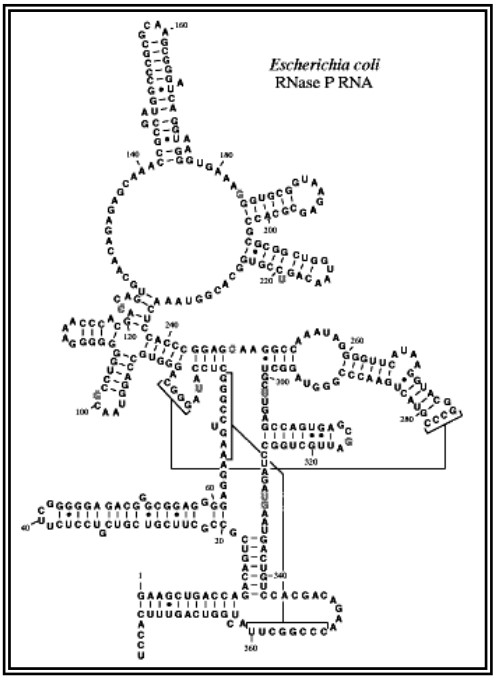

Figura 3: Plegamiento del ARNm de la RNase P *Escherichia coli* (Haas et al, 1996)

El estudio de la termodinámica de los vecinos más cercanos (“nearest neighbors”) en oligonucleótidos (Allawi y Santa Lucia, 1997), estableció un método unificado para el cálculo de la energía liberada por el plegamiento de secuencias de ARNm. La energía liberada depende del número y del tipo de bases involucradas en el plegamiento del ARNm. Es así como secciones plegadas ricas en guanina y citosina liberan considerablemente más energía, que secciones plegadas de igual tamaño ricas en timina y adenina.

El punto destacable y de mayor importancia en los antecedentes expuestos, es que estos han sido comprobados en todos los dominios de la vida. A esto se le suma la necesidad de una herramienta de bioinformática capaz de analizar y proponer mutaciones para un amplio espectro de organismos.

1.2.6 Herramientas bioinformáticas

La creciente necesidad de procesos de fabricación sustentable ha instaurado una tendencia a reemplazar los métodos tradicionales de síntesis química por métodos biotecnológicos para la elaboración de productos valiosos, tales como productos farmacéuticos, combustibles e ingredientes alimentarios. Una consecuencia directa de lo anterior es el aumento de la demanda de genes heterólogos con un alto nivel de expresión (Li y Bo, 2014). Esto, sin embargo, implica que el metabolismo de los microorganismos, por lo general, necesita ser modificado para cumplir con los propósitos de la industrial (Rocha et al, 2008). De este modo la expresión de proteínas funcionales en organismos hospederos heterólogos se ha transformado en la piedra angular de la biotecnología moderna.

En base a lo anterior, el campo de la ingeniería metabólica ha desarrollado herramientas para la introducción de modificaciones genéticas (Nielsen, 2001; Stephanopoulos et al, 1998). Primeramente, estas se han basado en principios de diseño cualitativo o intuitivo, sin adentrarse en modelos matemáticos efectivos que puedan predecir con precisión el comportamiento celular. Una de las aplicaciones más comunes es la adaptación de los codones usados en los genes al uso de codones específico del organismo hospedero heterólogo. En la forma de optimización más simple, cada aminoácido está representado por el codón sinónimo utilizado con mayor frecuencia en los genes altamente expresados del organismo hospedero. Sin embargo, se ha comprobado que estos procedimientos pueden conducir a la generación de sitios de restricción indeseados, limitaciones en la expresión, etc. (Raab et al, 2010). Sin duda, la mejor solución sería la de generar todas las combinaciones posibles de los codones que representan una secuencia de aminoácidos dada, y evaluar todos ellos con la ayuda de una función de calidad y finalmente elegir el que tenga la puntuación más alta de calidad. Por desgracia, el número de combinaciones posibles está en el rango de $10e^{47}$ incluso para una pequeña proteína de 100 aminoácidos, por lo que el enfoque descrito es imposible realizar en la práctica (Raab et al, 2010).

Diversos estudios han manipulado el uso de codones de una secuencia codificante en el intento de aumentar la eficiencia de traducción. Algunos han tenido éxito, mejorando la expresión de proteínas (Deng, 1997; Feng et al, 2000; Kotula y Curtis, 1991; Sinclair y Choy, 2002; Frelin et al, 2004), pero otros también, han fallado (Alexeyev y Winkler, 1999; Wu et al, 2004), lo que sugiere que el concepto de optimización de codones no es trivial.

Lo anterior, permite inferir que uno de los mayores desafíos en la ingeniería metabólica moderna es el desarrollo de modelos cuantitativos y algoritmos para identificar un conjunto de manipulaciones genéticas que resulten en una cepa microbiana con un

fenotipo metabólico deseable que normalmente significa, que tiene un alto rendimiento y productividad (Rocha et al, 2008).

Entre las herramientas de optimización disponibles que operan para aumentar el nivel de expresión de proteínas recombinantes, un número considerable utiliza el sesgo en el uso de codones como, prácticamente, único factor de optimización de secuencia, como por ejemplo OPTIMIZER (Puigbo et al, 2007), GeneDesign (Richardson et al, 2006) y Gene Designer (Villalobos et al, 2006). Otras herramientas han ido un paso más lejos como GeneGA (Li y Bo, 2014), que utiliza tanto el sesgo en el uso de codones como la estructura secundaria del ARNm para la optimización de genes.

Si bien, las herramientas mencionadas recientemente funcionan en su generalidad. Una herramienta que abarque un amplio espectro de genes y organismos hospederos, y que al mismo tiempo muestre un funcionamiento robusto y eficaz, aun no se encuentra disponible.

Si además se considera que la sustitución de codones no óptimos, representados por codones de la secuencia codificante que rara vez se utilizan en el organismo hospedero, por codones que corresponden a las especies más abundante de ARNt, puede aumentar considerablemente el rendimiento de la expresión heteróloga (Fuglsang, 2003).

En este contexto, se resolvió realizar un algoritmo para la optimización de codones de genes heterólogos que conjuga principalmente el sesgo en el uso de codones y la abundancia de ARNt en el organismo hospedero, además de la formación de estructuras secundarias del ARNm, como factores para guiar la mutación.

En base a los antecedentes descritos, el presente trabajo se focalizó en integrar y plasmar estas diferentes características en los algoritmos de un programa de bioinformática diseñado para el análisis y mutación de secuencias de ADN.

1.3 Objetivos

1.3.1 Objetivo general

Diseñar e implementar un algoritmo que integre diversos criterios experimentales en una interfaz gráfica, para el análisis de una secuencia de ADN codificante y sugiera mutaciones sinónimas para optimizar la expresión recombinante.

1.3.2 Objetivos específicos

1. Conformar una base de datos de frecuencia de uso de codones y abundancia de ARNt para diferentes divisiones de organismos.

2. Confeccionar una interfaz para el ingreso de un archivo contenedor de una secuencia de ADN y diseñar un algoritmo de detección automática del marco abierto de lectura.

3. Diseñar un algoritmo que cuantifique los codones del marco abierto de lectura y que además, detecte y asocie la secuencia a una base de datos.

4. Confeccionar dos interfaces para mostrar el análisis numérico y gráfico del marco abierto de lectura, y que además posibiliten la selección de bases de datos y la visualización de la información de la base de datos seleccionada.

5. Diseñar un algoritmo que clasifique y sugiera mutaciones sinónimas a los codones del marco abierto de lectura según la abundancia de ARNt correspondiente, la frecuencia de uso del codón en el genoma del organismo hospedero y la frecuencia de uso del codón en la secuencia codificante examinada.

6. Diseñar un algoritmo que identifique los codones con una baja abundancia de ARNt en la rampa inicial del marco abierto de lectura y que además, analice y posibilite la variación del tamaño de la rampa.

7. Confeccionar una interfaz para mostrar las mutaciones sugeridas al marco abierto de lectura y el análisis de la rampa, y que además permita establecer filtros adicionales a las mutaciones.

8. Diseñar un algoritmo que identifique la existencia y calcule la energía libre de las estructuras secundarias en el ARNm del marco abierto de lectura, y proponga mutaciones sinónimas capaces de evitar la formación de estructuras secundarias altamente estables.

9. Confeccionar una interfaz para el análisis de estructuras secundarias y que posibilite la visualización gráfica del plegamiento.

1.4 Metodología

El trabajo comenzó con una acuciosa revisión bibliográfica, de interiorización y recolección de información. La información recopilada se utilizó en la planificación del trabajo, que se subdividió en base al objetivo principal. Consecuentemente, las subdivisiones representan el desarrollo cronológico del trabajo para alcanzar dicho objetivo.

1.4.1 Armado y estructuración de las bases de datos

Las bases de datos en este trabajo representan la información de uso de codones y abundancia de ARNt de los organismos hospederos. Ambas bases de datos incluyeron, exactamente, las mismas especies y/o cepas. Cada base de datos se construyó y dispuso según divisiones (ver Tabla 3). Las divisiones se establecieron según la diversidad y disponibilidad de datos. La conformación de las divisiones en algunos casos incluyó: diferentes organismos, diferentes cepas de un mismo organismo, y en algunos casos, los datos provenientes de un solo organismo.

La base de datos de uso de codones se realizó, con los datos provenientes de la contabilización de cada codón, en todas las secuencias codificantes del ADN para cada uno de los organismos seleccionados (Kazusa DNA Research Institute, 2014). Análogamente, la base de datos de abundancia de ARNt se construyó con los datos provenientes de la predicción del número de copias de los genes de ARNt, en cada uno de los organismos (Genomic tRNA Database, 2014). Para las divisiones que incluyeron más de un organismo o cepa, los diferentes datos fueron ponderados de igual manera. De este modo y en consideración a los 64 codones existentes, a cada división se le fue calculado y asociado 64 valores fraccionales para la base de datos de uso de codones, y 64 valores fraccionales para la base de datos de abundancia de ARNt.

1.4.2 Entorno de programación

La extensión de las secuencias de ADN y la redundancia del código genético en términos de programación, se traducen en un gran número de datos, si a esto se le suma, análisis y mutaciones de secuencias de ADN, el número de datos aumenta considerablemente. Debido a esto, el diseño de la herramienta bioinformática de análisis y mutación de secuencias de ADN se realizó en el software de programación MATLAB, dado que éste opera bajo el funcionamiento básico de matrices y vectores, lo que facilita el manejo de grandes números de datos y la operación entre estos.

La programación se comenzó con la familiarización y estudio de los conceptos básicos de programación en MATLAB® versión R2013a. MATLAB permite el desarrollo de aplicaciones integradas y está orientado para proyectos con elevados cálculos matemáticos**.** La aplicación integra análisis numérico, cálculo matricial, proceso de señal y visualización gráfica en 2D y 3D. MATLAB tiene también, un lenguaje de programación propio, que permite desarrollar algoritmos y crear modelos y aplicaciones.

La escritura y formulación de algoritmos necesitó de la utilización del toolbox de bioinformática. Los toolboxes son programas de apoyos especializados de MATLAB, que incorporan nuevas funciones al programa principal.

El toolbox de bioinformática permite acceso a formatos de datos genómicos y proteomicos, técnicas de análisis y visualizaciones especializadas para secuencias genómicas y proteomicas. En el área de análisis de secuencias, que provee funciones para el secuenciamiento y visualización de secuencias genómicas y proteomicas, se utilizó la función para determinar la secuencia inversa complementaria de una secuencia. En el área de análisis estructural, específicamente en las funciones para la visualización y predicción de la estructura secundaria del ARN, se utilizó la función para graficar el plegamiento del ARN.

1.4.3 Implementación de interfaz con el usuario

La interfaz de un programa corresponde al punto de contacto o método de interacción entre el usuario y el programa de ordenador o computadora. El presente trabajo utilizó GUI de MATLAB para la programación de la interfaz. La interfaz gráfica de usuario (GUI) es una representación gráfica que contiene dispositivos o componentes, que permiten realizar tareas interactivas. Los componentes que hacen de GUI, una herramienta para la construcción de interfaces son:

Tipos de componentes:

- Botones:
  - Botón pulsador.
  - Botón de conmutación.
  - Botón de Radio.
  - Casilla de verificación.
- Listas:
  - Menú pop-up.
  - Cuadro de lista.
- Grupo de Botones.
- Paneles.
- Deslizadores.
- Ejes.

Las interfaces se confeccionaron de manera que resultasen amigables, y de fácil interacción y uso. De este modo, el usuario no requiere de conocimientos de programación para su utilización. La programación y conformación de la interfaz de la herramienta bioinformática de análisis y mutación de secuencias de ADN, se realizó en base a diferentes componentes. Se utilizaron 5 GUI (ventanas) consecutivas para el desarrollo parcializado de los objetivos del programa. Cada ventana se construyó con los componentes necesarios para transmitir y manipular de mejor manera la información dispuesta. La programación se desarrolló a través de metas semanales, las que avanzado el semestre dieron forma y completaron la herramienta.

1.4.4 Implementación de árboles de decisiones

Uno de los puntos de mayor importancia en el trabajo fue la determinación de las mutaciones. El ordenamiento de los criterios de mutación establecidos fue fundamental para la consistencia de las mutaciones. Para esto se realizó un árbol de decisión. Los árboles de decisiones son una de las herramientas más útiles y utilizadas para la toma de decisiones. Los arboles representan y categorizan una serie de condiciones que ocurren de forma sucesiva. Al utilizar un árbol, este siempre señalará un único camino dependiendo del valor de la variable evaluada.

El árbol de decisiones se construyó para clasificar a los codones según 3 valores, un primer valor fue la variable de ingreso, que corresponde al valor fraccional de presencia de un determinado codón en la secuencia o gen estudiado por el usuario. El segundo y tercer valor corresponden a los valores fraccionales para el mismo codón en las bases de datos de uso de codones y abundancia de ARNt. En disposición de estos valores, el árbol de decisiones califica al codón como: codón objeto de mutación con primera prioridad, codón objeto de mutación con segunda prioridad, codón objetivo para la mutación de otro codón con primera prioridad, codón objetivo para la mutación de otro codón con segunda prioridad y codón no objeto de mutación ni objetivo para la mutación de otro codón. Es decir clasifica al codón, para ser mutado o para ser el blanco de otras mutaciones, en diferentes grados, o en su defecto para no ser alterado de ninguna manera.

1.4.5 Estimación del plegamiento del ARNm

El plegamiento y unión de dos sectores no adyacentes del ARNm, puede afectar la traducción e incluso el plegamiento de la proteína. Es por esto que las mutaciones realizadas con el objeto de optimizar la expresión de una proteína, pueden resultar contraproducente si en el proceso forman secuencias inversamente complementarias.

A modo de evaluar el efecto de las mutaciones, se diseñó un análisis de plegamiento de la secuencia, posterior a las mutaciones en base al árbol de decisiones. El análisis consistió en una identificación de secuencias y sus respectivas secuencias inversas complementarias, a las que se le calculó la diferencia de energía libre de Gibbs ΔG^o. A menor valor de ΔG^o, mayor es la estabilidad del plegamiento y más probable es que afecte a la traducción. El valor del ΔG^o se obtuvo mediante el cálculo de vecinos más cercanos en oligonucleótidos.

Adicional al análisis, se dispuso la posibilidad de mutar codones dentro de las secuencias plegadas, siguiendo el mismo árbol de decisiones antes expuesto, con la excepción que para estas mutaciones se permiten todo tipo codones, priorizando las mutaciones convenientes por sobre las no convenientes y riesgosas Esto de acuerdo a las clasificaciones realizadas por el árbol de decisiones.

1.4.6 Evaluación de resultados

La evaluación de las mutaciones realizadas bajo el diseño del árbol de decisiones se realizó mediante la comparación de las secuencias mutadas de los genes eGFP y ADN polimerasa en contraste a las secuencias mutadas de estos mismos genes realizadas por otros trabajos, que cuentan con la comprobación empírica del mejoramiento en el expresión.

Para la comparación se realizó, un test de hipótesis o prueba de significación, que en este caso particular, es el procedimiento para juzgar si las mutaciones realizadas para la optimización de la expresión de un gen son compatibles con lo observado en las mutaciones realizadas en este mismo gen, pero empíricamente comprobadas como beneficiosas para la expresión. Mediante esta teoría, se aborda el problema estadístico considerando una hipótesis determinada H_0 y otra hipótesis H_1, y se intenta dirimir el grado verdad en ambas hipótesis. Esta operación está asociada a los errores de tipo I y II, que definen respectivamente, la posibilidad de tomar un suceso falso como verdadero, o uno verdadero como falso

II Desarrollo

2.1 Bases de datos de uso de codones y abundancia de ARNt

En la conformación de las 2 bases de datos, se incluyeron las mismas especies y/o cepas. Cada base de datos se construyó y dispuso según divisiones (ver Tabla 2). Las divisiones se establecieron según la diversidad y disponibilidad de datos (Kazusa DNA Research Institute, 2014; Genomic tRNA Database, 2014). Para el caso de *Escherichia coli* se ponderaron las diferentes cepas y para los casos de insectos dípteros y plantas angiospermas monocotiledóneas se ponderaron las diferentes especies a fin de caracterizar la división (ver Anexos 5.2).

Nº	Divisiones	Especies/Cepa
1	*Escherichia coli*	536
		APEC O1
		CFTO73
		K12
		O157H7
		O157H7EDL933
		UTI89
2	*Bacillus subtilis*	-
3	*Saccharomyces cerevisiae*	S288c
4	*Cricetulus griseus*	CHO+K1
5	*Insecta diptera*	*Anopheles gambiae*
		Drosophila melanogaster
6	*Homo sapiens*	-
	Plantas Embryophytas:	
7	Bryophyta	*Physcomitrella patens*
8	Angiosperma dicotiledóneas	*Vitis vinifera*
9	Angiosperma monocotiledóneas	*Brachypodium distachyon*
		Sorghum bicolor
		Zea mays

Tabla 2: División bases de datos

2.2 Criterios de Mutación

Como se mencionó anteriormente, se construyó un árbol de decisiones para catalogar a los codones de la secuencia o gen estudiado. Los criterios de clasificación de codones estuvieron basados en los resultados y conclusiones de diversos estudios, los que por regla general señalan a la abundancia de ARNt como factor preponderante, y por sobre el uso de codones, para la mutación y optimización de genes recombinantes. Los criterios de clasificación fueron los siguientes.

El primer nivel de decisión se subdividió según el valor de la variable ingresada, en este caso el valor fraccional del codón analizado en la secuencia o gen estudiado. Las subdivisiones fueron: de 0% a 1%, de 1% a 2% y mayores a 2%. En un segundo nivel

de decisión se subdividió según el valor de abundancia de ARNt. Mientras que el tercer nivel de decisión se subdividió según el uso de codones. Ambos, segundo y tercer nivel se subdividieron en los mismos rangos que el primer nivel. Los porcentajes de estos rangos fueron elegidos y determinados arbitrariamente después de un análisis visual y practico de los porcentajes en el uso de codones de diferentes genes. Después del tercer nivel de decisión, se estableció un primer nivel de acontecimiento, que clasifica los codones que tienen valores de uso de codones y abundancia de ARNt similares. Para los casos restantes, se estableció un cuarto nivel de decisión, que es utilizado cuando los valores de uso de codones y de abundancia de ARNt se diferencian considerablemente. En estos casos se realizó una ponderación de 75% hacia la abundancia de ARNt y de un 25% al uso de codones, esto debido las razones antes expuestas. Finalmente para terminar, se dispuso de un segundo nivel de acontecimiento, que clasifica la totalidad de codones restantes.

Otras consideraciones adicionales, resultaron producto que se consideró a la abundancia de ARNt como un factor preponderante. Esto se reflejó en que ante, prácticamente, cualquier escenario en que la abundancia de ARNt estuviera en el rango de 0% a 1%, el codón analizado era catalogado para ser mutado. Otro punto donde se evidencia la importancia que se le otorgó a la abundancia de ARNt, ocurre en casi la totalidad de casos, y consiste en permitir un porcentaje mayor de frecuencia del codón en la secuencia analizada (+0,5%), con respecto al porcentaje de abundancia de ARNt para ese mismo codón.

Bajo estos criterios se clasificó a los codones de la secuencia o gen analizado como: codón objeto de mutación con primera prioridad, codón objeto de mutación con segunda prioridad, codón objetivo para la mutación de otro codón con primera prioridad, codón objetivo para la mutación de otro codón con segunda prioridad y codón no objeto de mutación ni objetivo para la mutación de otro codón. Estas clasificaciones se utilizan para la selección de codones a mutar y del codón sinónimo por cuál se lo muta.

2.3 Plegamiento del ARNm

En el marco de este trabajo, se realizó un cálculo de la energía involucrada en el plegamiento del ARNm. Esta energía libre se estimó mediante la termodinámica de los vecinos más cercanos (“nearest neighbors”) en oligonucleótidos (Santa Lucia, 1997). Lo anterior permitió el cálculo predictivo del $\Delta G°$ de las secuencias complementarias (ver ejemplo de cálculo en Anexos 5.1). En esta estimación, el $\Delta G°$ total, a una temperatura ambiente de 37°C, está dado por:

$$\Delta G°_{37}(total) = \sum_i n_i \, \Delta G°(\mathrm{i}) + \Delta G°(\text{iniciación con el terminal G} * \mathrm{C}) + \Delta G°(\text{iniciación con el terminal A} * \mathrm{T}) + \Delta G°(\text{simetría})$$

en donde los $\Delta G°(i)$ son los cambios de la energía libre estándar de los 10 posibles vecinos Watson-Crick más cercanos (ver tabla 3), los n_i son el número de ocurrencias de cada vecino más cercano y $\Delta G°(simetría)$ es el costo de mantención de la simetría en secuencias auto-complementarias. El valor de este último parámetro es de 0,43

[kcal/mol] para secuencias auto-complementarias y cero para secuencias no auto-complementarias (Santa Lucia, 1997).

Secuencia	ΔH°	ΔS°	ΔG°
	[kcal/mol]	[cal/k*mol]	[kcal/mol]
AA/TT	-7,9	-22,2	-1,00
AT/TA	-7,2	-20,4	-0,88
TA/AT	-7,2	-21,3	-0,58
CA/GT	-8,5	-22,7	-1,45
GT/CA	-8,4	-22,4	-1,44
CT/GA	-7,8	-21,0	-1,28
GA/CT	-8,2	-22,2	-1,30
CG/GC	-10,6	-27,2	-2,17
GC/CG	-9,8	-24,4	-2,24
GG/CC	-8,0	-19,9	-1,84
Inicio/termino G C	0,1	-2,8	0,98
Inicio/termino A T	2,3	4,1	1,03

Tabla 3: Parámetros unificados de vecinos más cercanos ΔH°, ΔS° y ΔG° en NaCl 1[M] (Allawi y Santa Lucia, 1997)

2.4 Implementación computacional

La herramienta de análisis de secuencias de ADN, se implementó en 5 etapas (ver código en Anexos 5.3), cada uno de las cuales aporta criterios para sugerir mutaciones sinónimas.

2.4.1 Primera Etapa: Ingreso del archivo de secuencia

La primera etapa de desarrollo fue la implementación de una ventana, para el ingreso del archivo, donde se encuentra almacenada la secuencia a analizar (ver Figura 4).

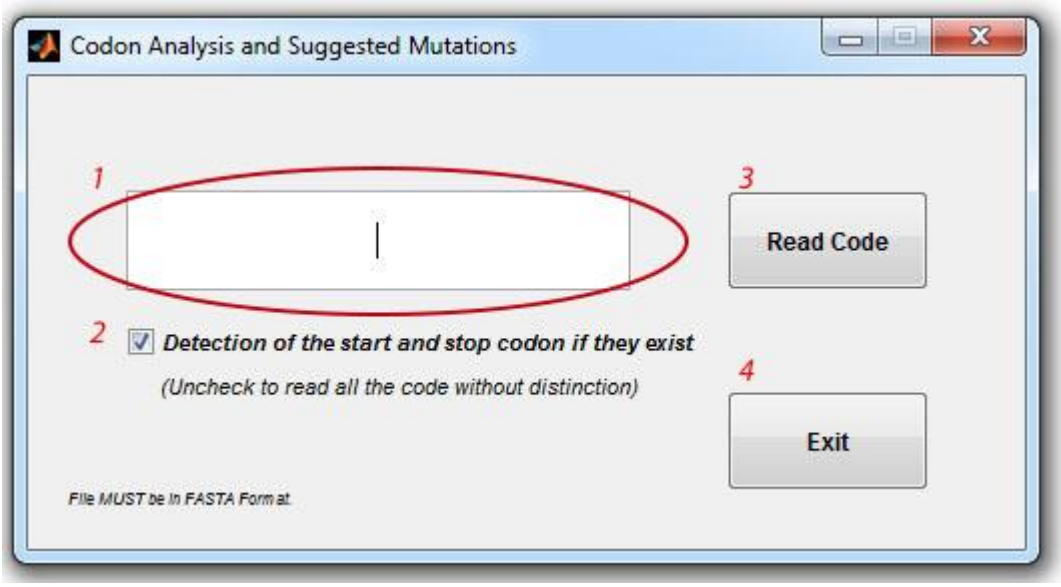

Figura 4: Ventana 1: Ingreso del archivo de secuencia

En la figura pueden verse los siguientes elementos que están incluidos en la ventana de interfaz con el usuario.

1. Cuadro para ingresar el nombre y extensión del archivo.
2. Check box para la identificación automática de codones de inicio y termino (activado por defecto).
3. Botón de lectura del código en el archivo ingresado.
4. Botón de salida del programa.

El archivo que contiene la secuencia de ADN a analizar debe estar en formato FASTA para una correcta recuperación del código almacenado.

Una vez ingresado el nombre de archivo, el software lo lee y procede a identificar el marco abierto de lectura más probable de la secuencia. Para esto, se realiza una identificación automática de codones de inicio y término en el código recuperado del archivo. El programa filtra y rescata la secuencia *codón de inicio-codón de término* más larga presente. Esta implementación también maneja casos especiales. Cuando se identifica codones de inicio sin codones de término, se entrega la secuencia *primer codón de inicio- fin del código.* En el caso de no encontrar codones de inicio, se entrega

la secuencia del código sin modificación. Al desactivar el check box el programa opera de igual modo que en este último caso.

La activación o no del check box tiene repercusiones en el análisis posterior. Al activar el check box se infiere la presencia de un gen con inicio y término. Al trabajar sobre un gen, el programa tiene la particularidad de considerar y asumir la presencia de rampas, que corresponden al sector cercano al inicio. Dicha zona tiene reglas particulares de mutación y es estudiada en las siguientes etapas. Por el contrario al desactivar el check box se asume que se trabaja sobre una secuencia de ADN sin distinciones mayores, y se obvia la diferenciación y análisis relacionado a las rampas.

2.4.2 Segunda Etapa: Análisis de la secuencia

La segunda etapa de desarrollo consistió en la implementación de un programa con una interfaz de tipo ventana, para la detección y asociación automática del código a una de las divisiones establecidas en la base de datos, tanto de uso de codones como de abundancia de ARNt. Además, en esta ventana se ilustra el uso de codones de la secuencia introducida (ver Figura 5).

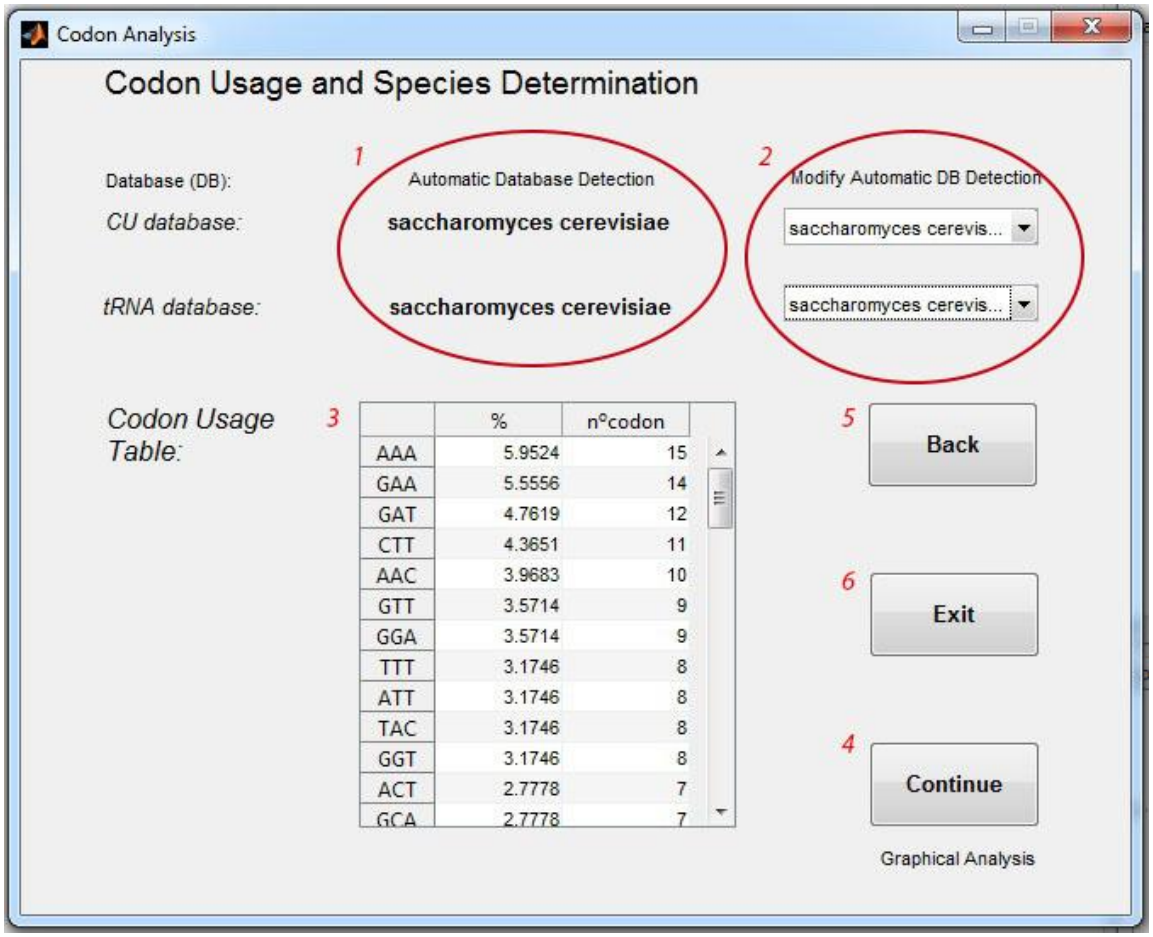

Figura 5: Ventana 2: Análisis de la secuencia introducida

En la figura de la interfaz se pueden apreciar los siguientes componentes:

1. Detección y asociación automática del código a una de las divisiones de la base de datos.
2. Menús emergentes para modificar la asociación a una diferente división de las bases de datos.
3. Tabla de uso de codones.
4. Botón para continuar al análisis gráfico.
5. Botón para volver a la ventana anterior.
6. Botón de salida del programa.

En la tabla de uso de codones se detalla en orden descendente de frecuencia los codones con mayor presencia, indicando el porcentaje y el número asociado.

La detección y asociación automática del código se realiza mediante una comparación de la suma de las varianzas, entregadas por el cálculo de la varianza codón por codón entre el uso de codones de la secuencia estudiada y las bases de datos de uso de codones y abundancia de ARNt de las diferentes divisiones. De este modo, se obtiene un varianza total para cada una de las divisiones en ambas bases de datos. El menor valor de varianza total representa e indica la división, que mejor se asemeja a la secuencia estudiada.

Esta automatización entrega al usuario la posibilidad de conocer a qué división, y por lo tanto, a qué especie o especies, se asemeja o se adaptaría de mejor manera el código. Adicionalmente, si la división establecida automáticamente no es la requerida, los menús emergentes permiten modificar la división según prefiera el usuario.

En las etapas siguientes, las bases de datos escogidas se utilizan como puntos de comparación y apoyo para la manipulación del código.

2.4.3 Tercera Etapa: Análisis grafico de la secuencia

La tercera etapa de desarrollo fue la implementación de un programa y una interfaz de ventana para el análisis grafico de la secuencia (ver Figura 6 y Anexos 5.4).

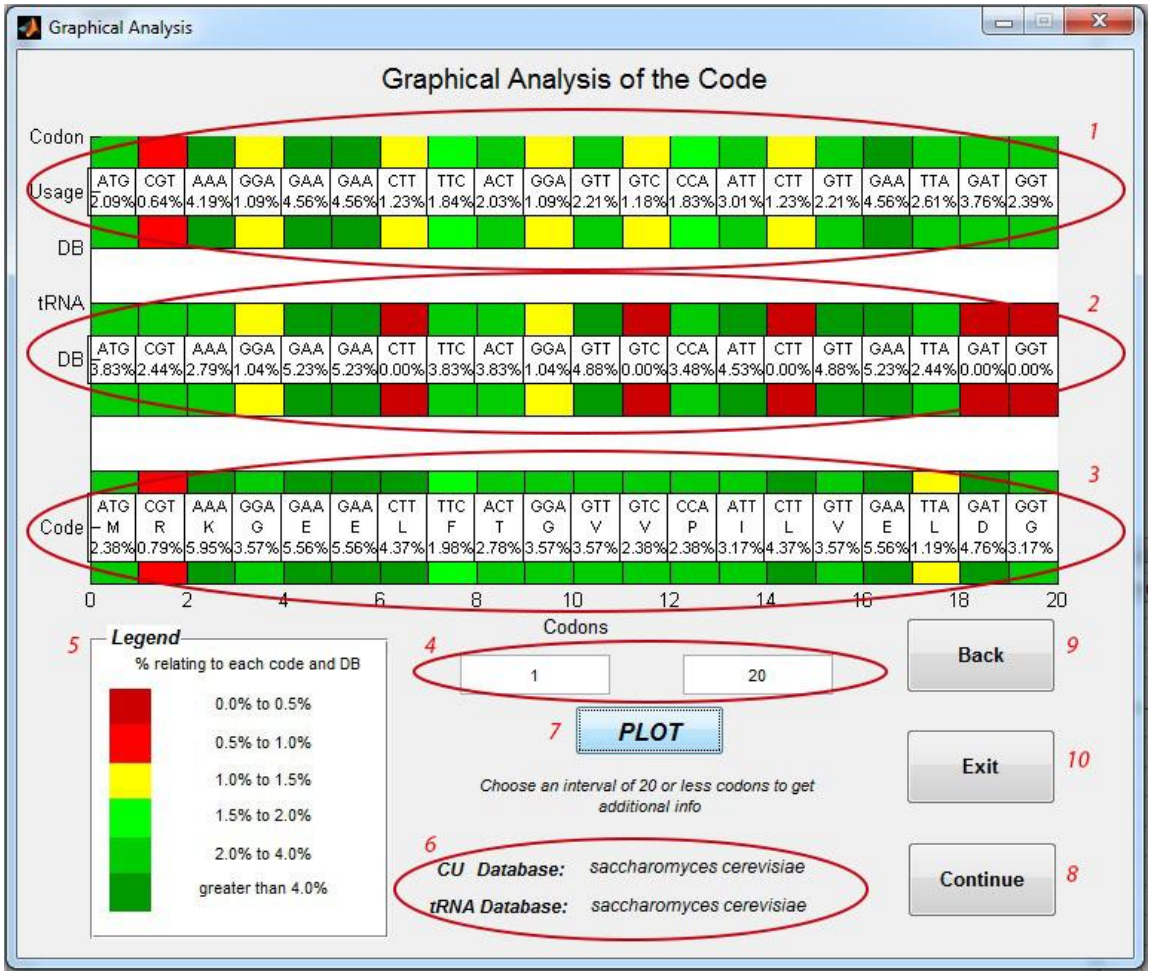

Figura 6: Ventana 3: Análisis gráfico detallado de la secuencia

El detalle de la disposición de la información y funciones ejemplificadas en la figura es la siguiente:

1. Grafico- datos de la base de uso de codones.
2. Grafico- datos de la base de abundancia de ARNt.
3. Grafico- datos de la secuencia.
4. Cuadros para establecer intervalo de graficado.
5. Leyenda.
6. Bases de datos utilizadas.
7. Botón para modificar el área de graficado.
8. Botón para continuar a mutaciones sugeridas.
9. Botón para volver a la ventana anterior.
10. Botón de salida del programa.

El análisis grafico permite apreciar la secuencia del código en su totalidad, estructurada en codones coloreados según su frecuencia en el código. Lo anterior permite identificar visualmente zonas dentro del código con presencia de codones raros o de codones abundantes. Adicionalmente, se grafica la secuencia del código con los datos de las bases de datos de uso de codones y abundancia de ARNt. Esto permite al usuario establecer un nexo visual y comparar el uso de codones de la secuencia con los usos de codones y abundancia de ARNt de la división escogida, representada en las bases de datos.

En detalle, el gráfico en la interfaz está compuesto por 3 gráficos independientes, el primer grafico (señalado con el número 1 en la figura 6) corresponde a la frecuencia del uso de codones de la división escogida, representada por base de datos de uso de codones. El segundo gráfico (señalado con el número 2 en la figura 6) corresponde a la abundancia de ARNt para cada codón, de acuerdo a datos de la base de datos de abundancia de ARNt escogida. Por último, el tercer gráfico (señalado con el número 3 en la figura 6) corresponde a la frecuencia del uso de codones en la secuencia analizada. Los colores y porcentajes señalan frecuencia en el uso de codones o abundancia de ARNt, según corresponda. Los colores se dispusieron de acuerdo a los valores señalados en la leyenda (número 5 en la figura 6).

Complementariamente, está la opción de graficar intervalos o sectores de la secuencia. Al graficar un intervalo menor a 20 codones, la gráfica muestra información adicional, que incluye el codón, el aminoácido codificado y el valor porcentual de la frecuencia de cada codón en la secuencia analizada, además de los valores porcentuales del uso de codones y abundancia de ARNt de las respectivas bases de datos para cada codón.

2.4.4 Cuarta Etapa: Mutaciones sugeridas

La cuarta etapa de desarrollo fue la implementación de un programa de análisis y de interfaz gráfica tipo ventana, para presentar las mutaciones sugeridas y el análisis de la rampa (ver Figura 7 y Anexos 5.4).

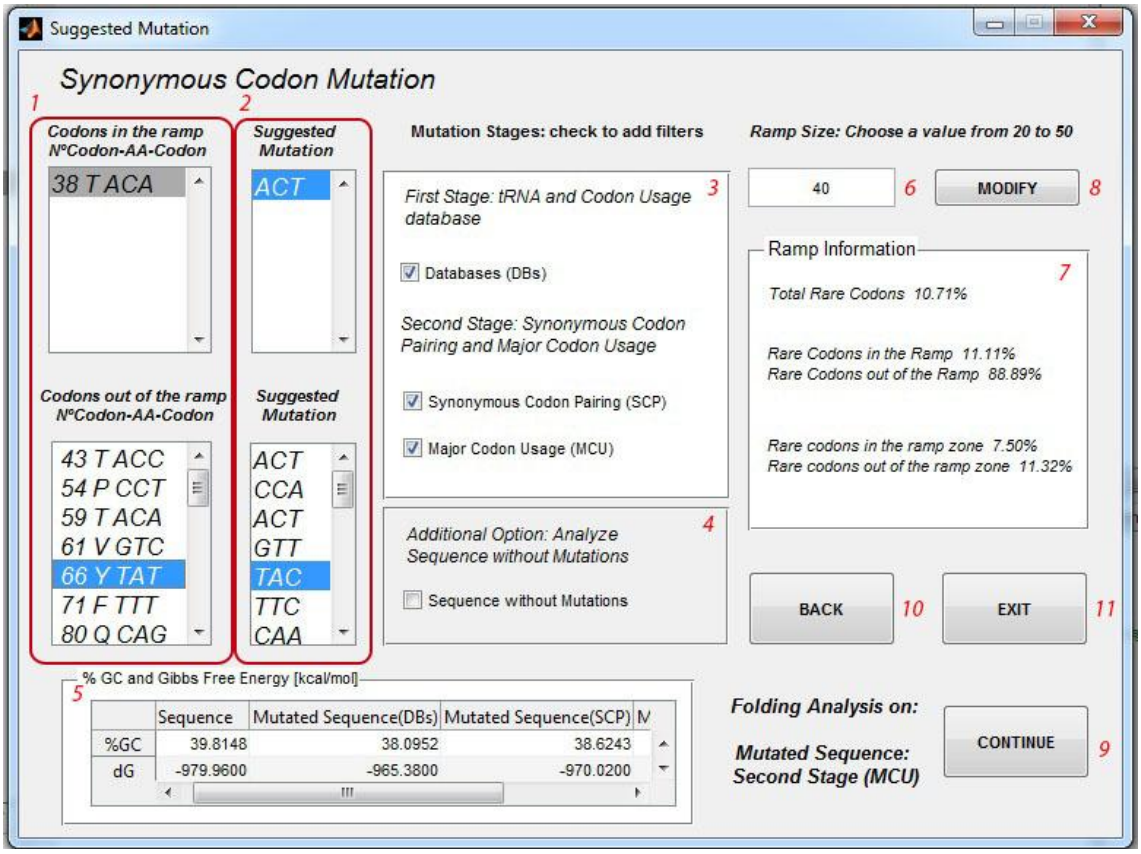

Figura 7: Ventana 4: Mutaciones sugeridas con Rampa

Los componentes, opciones y funciones descritos en la figura son los siguientes:

1. Lista de codones del código a mutar (numero/aminoácido/codón).
2. Lista de codones sugeridos como mutación.
3. Check box para adicionar filtros de mutación.
4. Check box para seleccionar el código sin mutaciones.
5. Tabla de energía libre de Gibbs y %GC.
6. Cuadro para modificar el tamaño de la rampa.
7. Cuadro de análisis de la rampa.
8. Botón para modificar el sector de la rampa.
9. Botón para continuar al análisis de plegamiento.
10. Botón para volver a la ventana anterior.
11. Botón de salida del programa.

En la ventana se presentan las sugerencias de mutación. Esto se realiza mediante una lista que identifica el número del codón, el aminoácido y el codón objetivo y/o candidato para mutación, y una segunda lista se disponen los codones sinónimos sugeridos para realizar la mutación. En el caso en que se esté analizando un gen (es decir, cuando el

check box de la etapa 1 está activado), las listas se dividirán en dos, estableciendo una zona de rampa y otra fuera de la rampa. Como se mencionó anteriormente, la zona de rampa es analizada y mutada bajo los parámetros descritos en los antecedentes, que resultan en una disminución de las mutaciones en la zona.

El tamaño de rampa puede ser modificado de acuerdo a las características de procedencia del código de la secuencia o a la preferencia del usuario. A modo de complementar la información se presenta en la ventana un cuadro, que detalla las características de la rampa en la secuencia.

Por defecto se presentan las mutaciones sugeridas derivadas de las bases de datos. Para identificar las mutaciones más convenientes se encuentran los check box con filtros adicionales. Al activarse, se filtran las mutaciones que cumplen las propiedades activadas. El activar o desactivar los check boxes determina también, la secuencia que se analiza en la siguiente etapa. Adicionalmente, existe la opción de activar un check box para seleccionar la secuencia del código sin mutar para el análisis de la siguiente etapa.

Como información previa al análisis de plegamiento de la siguiente etapa, se incluyó la tabla de energía libre de Gibbs (ΔG^o) y de porcentaje de guanina y citosina (%GC) para todas las posibles secuencias que son resultado de los diferentes filtros de mutación, además de la secuencia del código sin mutar como referencia.

2.4.5 Quinta Etapa: Análisis de plegamiento

La quinta etapa de desarrollo consistió en la implementación de un programa y una interfaz de ventana para el análisis de plegamiento de la secuencia de ARN mensajero (ver Figura 8).

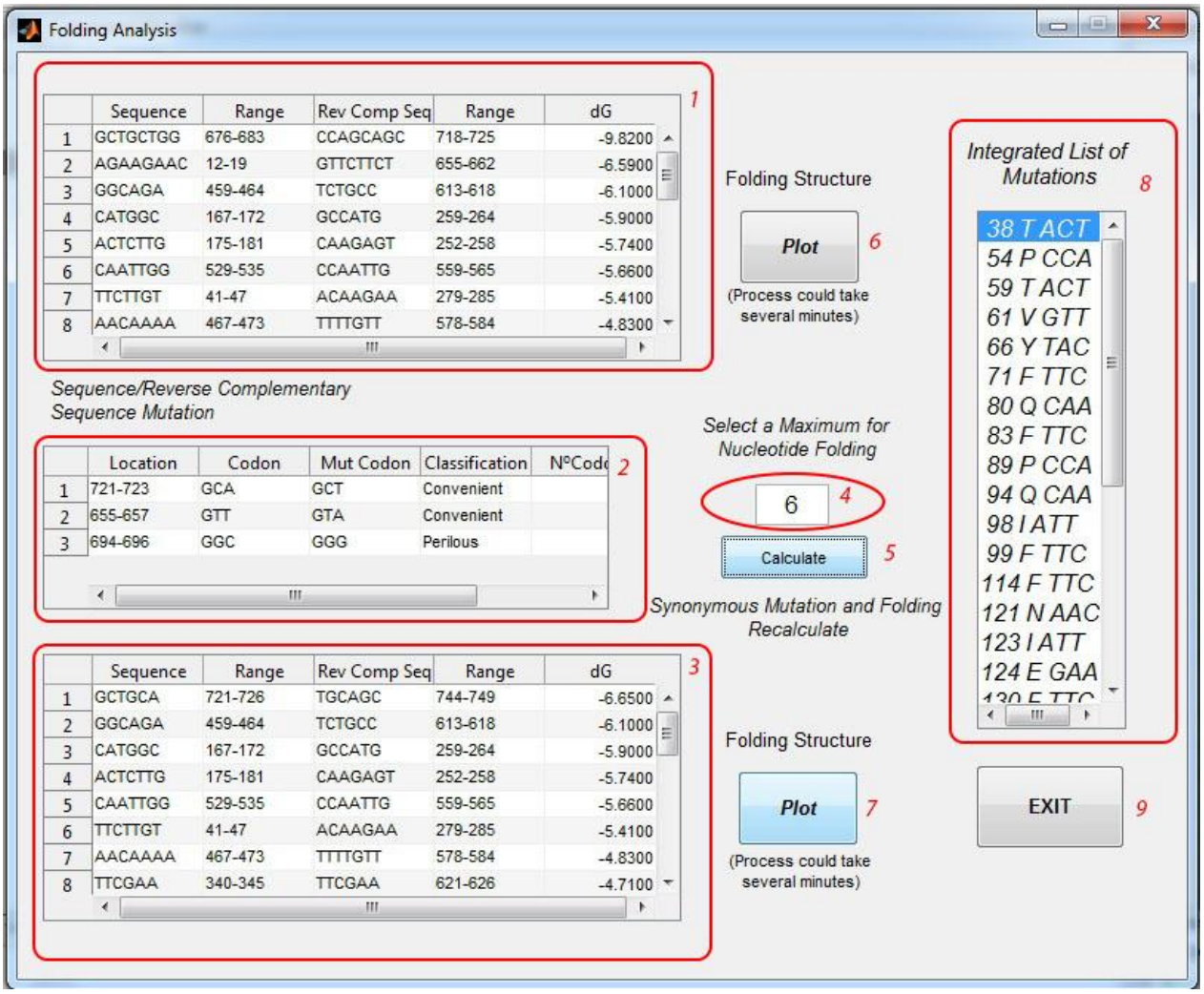

Figura 8: Ventana 5: Análisis de plegamiento del ARNm

Los componentes y la disposición de la información en la figura, es la siguiente:

1. Tabla de plegamiento de la secuencia.
2. Tabla de mutaciones.
3. Tabla de plegamiento de la secuencia después de ser mutada.
4. Cuadro para ingresar restricción al plegamiento.
5. Botón para mutar y calcular el plegamiento del código resultante.
6. Botón para graficar el plegamiento de la secuencia.
7. Botón para graficar el plegamiento de la secuencia después de ser mutada.
8. Lista de unificada de mutaciones.
9. Botón de salida de la ventana.

El análisis de plegamiento se presenta en la tabla de plegamiento de la secuencia analizada, en donde se identifican las secuencias, las secuencias reversas complementarias, los respectivos intervalos numéricos, asociados al número ordinal del nucleótido dentro de la secuencia, y los valores de $\Delta G°$ correspondiente. Este último parámetro, se utiliza para el ordenamiento descendente de la tabla de plegamiento de la secuencia.

Para modificar el plegamiento se dispone de un cuadro, para que el usuario ingrese una restricción al plegamiento en la secuencia (extensión del plegamiento en número de nucleótidos). Si se supera esta restricción se muta el código y se recalcula el plegamiento.

La tabla de mutaciones indica la locación según el número de nucleótido, el codón objeto de la mutación, el codón sinónimo al que se mutó, la clasificación de la mutación, el número de codón y el aminoácido al que corresponde. La clasificación de la mutación corresponde al grado de conveniencia de la mutación, y está directamente relacionada al codón sinónimo al que se mutó. La tabla de plegamiento de la secuencia después de ser mutada, muestra el plegamiento del código bajo las restricciones impuestas por el usuario.

La restricción al plegamiento consiste en establecer un límite al largo, y por lo tanto, al número de nucleótidos que pueden estar plegados. El algoritmo se diseñó para operar sobre los plegamientos, que resultaron del primer análisis de plegamiento y que están descritos en la tabla de plegamiento de la secuencia. Sin embargo, al eliminar plegamientos, esto posibilita la aparición de nuevos plegamientos que estaban anteriormente limitados por plegamientos que contribuían mayormente a la estabilidad del ARNm. Para solucionar este problema, el algoritmo una vez mutado y limitado los plegamientos existentes, recalcula el plegamiento y vuelve a operar sobre los plegamientos que aparezcan. Los resultados se disponen en la tabla de mutaciones y en la tabla de plegamiento de la secuencia después de ser mutada.

Para complementar el análisis, se dispone de la opción para graficar la secuencia antes y después de ser mutada (ver Figura 9), a fin de facilitar la visualización del nuevo plegamiento.

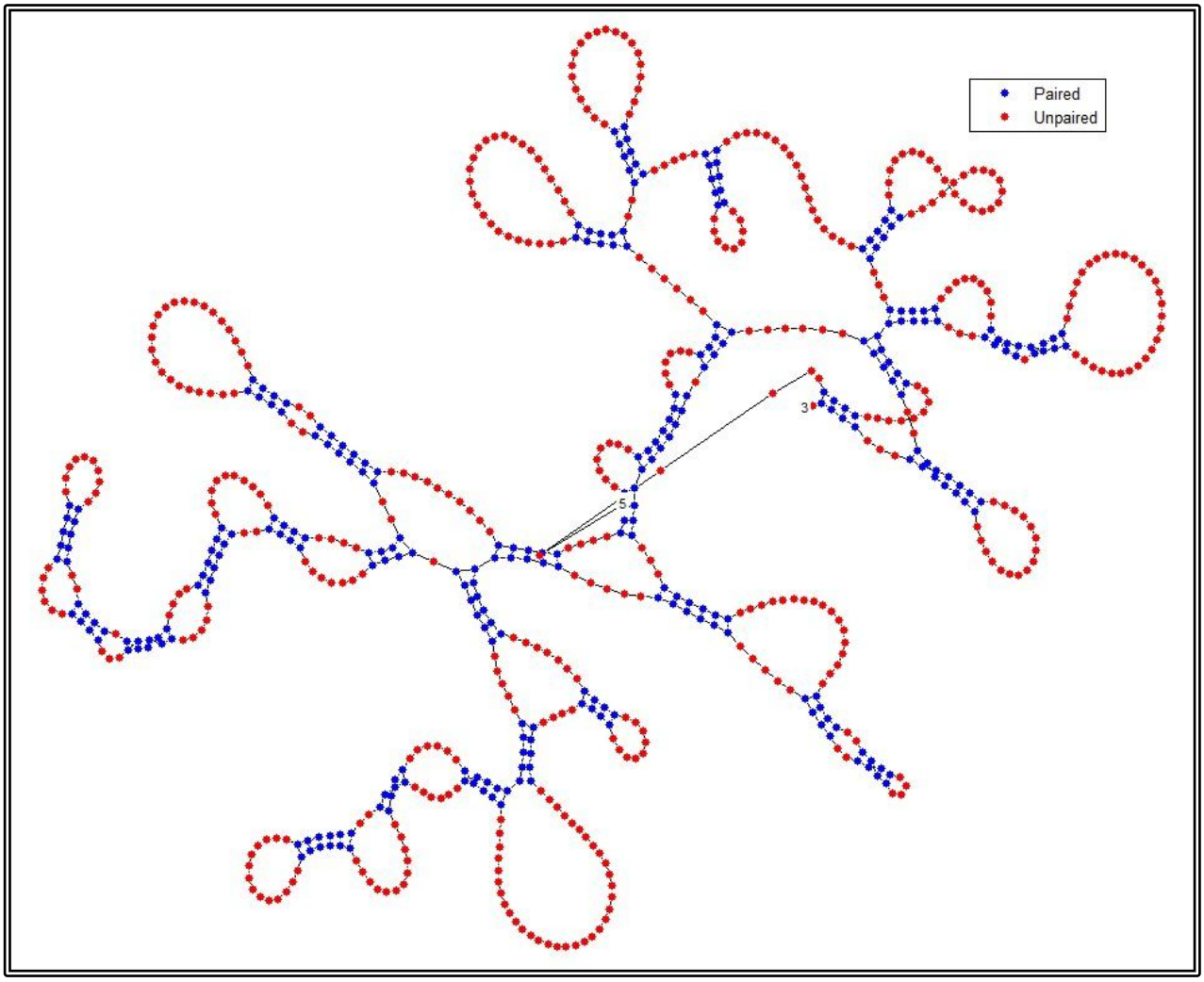

Figura 9: Ilustración del plegamiento del gen eGFP tipo silvestre

2.5 Validación de las sugerencias de mutación

La expresión de genes heterólogos es una herramienta importante para la biología sintética que permite la ingeniería metabólica y la producción de productos biológicos no naturales en una variedad de organismos hospederos (Lanza et al, 2014).

La eficiencia de traducción de genes heterólogos a menudo puede mejorarse mediante la optimización del uso de codones sinónimos a modo de adaptación al organismo hospedero. La optimización de codones se utiliza comúnmente para mejorar la expresión de genes heterólogos, especialmente en el contexto de la biología sintética y la ingeniería metabólica y celular (Lanza et al, 2014). La optimización de codones es una importante herramienta de la biología sintética que permite la expresión de ADN recombinante.

Para la validación de las mutaciones sugeridas por la herramienta bioinformática de análisis de secuencias de ADN, se realizó un contraste de hipótesis. Se abordó el problema estadístico considerando una hipótesis determinada H_0 y otra hipótesis H_1, y

se evaluó el grado veracidad en ambas hipótesis. Esta operación está asociada a los errores de tipo I y II.

H_0: Codones en la secuencia del gen mutados por la herramienta bioinformática de análisis de secuencias de ADN.
H_1: Codones en la secuencia del gen no mutados por la herramienta bioinformática de análisis de secuencias de ADN.

En un estudio de investigación, el error de tipo I también denominado error de tipo alfa (α) o falso positivo, es el error que se comete cuando se acepta Ho siendo esta falsa en el gen comprobado empíricamente. Esto es equivalente a encontrar un resultado falso positivo, porque se llega a la conclusión de que se ha declarado una mutación donde no existe en la realidad. El error de tipo II, también llamado error de tipo beta (β) o falso negativo, se comete cuando se acepta H_1 siendo esta falsa en el gen comprobado empíricamente. Esto es equivalente a la probabilidad de un resultado falso negativo, ya que se llega a la conclusión de que se ha sido incapaz de encontrar una mutación que existe en la realidad

En consideración a las dos hipótesis, se trabajó sobre los genes eGFP y ADN polimerasa. Ambos genes fueron analizados por la herramienta bioinformática de análisis y mutación de secuencias de ADN, descrita en el presente trabajo. Las secuencias de los genes mutados fueron evaluadas, en comparación, por las secuencias mutadas de los mismos genes pero con comprobación empírica de su incremento en la eficiencia de expresión. Estas mutaciones califican y establecen la veracidad o falsedad de las mutaciones realizadas por la herramienta del presente trabajo. De las comparaciones, se obtener cuatro posibles situaciones: Falso Positivo, Falso Negativo, Verdadero Positivo y Verdadero Negativo.

Para lo anterior se dispuso de secuencias de genes mutados con comprobada, alta expresión (Lanza et al, 2014; Welch et al, 2009). Estos mismos genes pero de tipo salvaje fueron analizados por la herramienta bioinformática descrita en el presente trabajo. Los resultados definieron las hipótesis para el análisis. Bajo estas hipótesis se analizó las mutaciones sugeridas frente a los resultados obtenidos por trabajos experimentales en los que se caracterizó la expresión de proteínas recombinantes de tipo silvestre y mutantes.

En un primer caso particular, se estudió la de optimización de codones en el gen heterólogo, que codifica la proteína fluorescente eGFP expresada en *Saccharomyces cerevisia,* en el trabajo de Lanza y cols. (2014). Se utilizó las versiones mutadas de eGFP *high expression table* (HT)y de eGFP *control matrix* 2 (C2), asociadas con la más alta y baja expresión respectivamente (Lanza et al, 2014). Estas mutaciones fueron comparadas con los codones de la secuencia de eGFP mutados por la herramienta del presente trabajo (ver secuencias en Anexos 5.5.1). Los resultados del análisis se presentan a continuación.

		Mutaciones sugeridas en eGFP	
eGFP HT		H_0	H_1
	H_0	95,2%	49,0%
		(Verdadero Positivo)	(Falso Negativo)
	H_1	4,8%	51,0%
		(Falso Positivo)	(Verdadero Negativo

Tabla 4: Validación mutaciones gen eGFP (alta expresión)

		Mutaciones sugeridas en eGFP	
eGFP C2		H_0	H_1
	H_0	73,8%	46,7%
		(Verdadero Positivo)	(Falso Negativo)
	H_1	26,2%	53,3%
		(Falso Positivo)	(Verdadero Negativo

Tabla 5: Validación mutaciones gen eGFP (baja expresión)

Si bien los valores de verdaderos positivos son altos al comparar con el gen altamente expresado, el alto porcentaje de falsos negativos no es un buen indicador para la validación de la herramienta bioinformática desarrollada.

Adicionalmente, se desglosó los verdaderos positivos, en términos de tipo de mutación, para eGFP HT y eGFP C2. Se catalogó como mutación sinónima idéntica cuando, por ejemplo, un particular codón del gen eGFP HT está mutado por un determinado codón sinónimo, y a su vez, este mismo particular codón fue mutado en el gen eGFP analizado por la herramienta bioinformática y además, por el mismo codón sinónimo que en el gen eGFP HT. En el caso que la herramienta bioinformática mute pero por un diferente codón sinónimo, el tipo de mutación se catalogó como mutación sinónima no idéntica.

	Tipo mutación	Mutaciones sugeridas en eGFP
eGFP HT	Mutación Sinónima idéntica	55,0%
	Mutación Sinónima no idéntica	45,0%
eGFP C2	Mutación Sinónima idéntica	35,5%
	Mutación Sinónima no idéntica	64,5%

Tabla 6: Desglose Verdaderos Positivos eGFP

Se observó una disminución de mutaciones sinónimas idénticas entre la comparación de las mutaciones en el gen eGFP HT con las mutaciones sugeridas en el gen eGFP por la herramienta bioinformática y la comparación de mutaciones en el gen eGFP C2 con las mutaciones sugeridas en el gen eGFP por la herramienta bioinformática

Para una segunda validación de la herramienta desarrollada en este trabajo, se escogió como objetivo, la exploración sistemática del efecto del uso de codones sinónimos en la expresión del gen codificante de la ADN polimerasa del fago Φ 29 de Bacillus (Blanco y

Salas, 1984). Este gen se seleccionó debido a que codifica evolutiva, estructural y funcionalmente a diferentes proteínas. Se analizaron las versiones mutadas de ADN polimerasa P19, P15, con la más alta y baja expresión respectivamente (Welch et al, 2009), y la secuencia de la ADN polimerasa mutada por la herramienta del presente trabajo (ver secuencias en Anexos 5.5.2). Los resultados del análisis se presentan a continuación.

		Mutaciones sugeridas en Polimerasa	
Pol P19		H_0	H_1
	H_0	98,8%	93,5%
		(Verdadero Positivo)	(Falso Negativo)
	H_1	1,2%	6,5%
		(Falso Positivo)	(Verdadero Negativo

Tabla 7: Validación mutaciones gen ADN Polimerasa (alta expresión)

		Mutaciones sugeridas en Polimerasa	
Pol P15		H_0	H_1
	H_0	80,0%	54,8%
		(Verdadero Positivo)	(Falso Negativo)
	H_1	20,0%	45,2%
		(Falso Positivo)	(Verdadero Negativo

Tabla 8: Validación mutaciones gen ADN Polimerasa (baja expresión)

Se observó un altísimo porcentaje de verdaderos positivos, y al igual que en el caso anterior, un alto porcentaje de falsos negativos, que nuevamente no benefician la validación de la herramienta bioinformática.

Adicionalmente, se desglosó los verdaderos positivos, en términos de tipo de mutación, para Pol P19 y Pol P15.

	Tipo mutación	Mutaciones sugeridas en Polimerasa
Pol P19	Mutación Sinónima idéntica	54,4%
	Mutación Sinónima no idéntica	45,6%
Pol P15	Mutación Sinónima idéntica	45,3%
	Mutación Sinónima no idéntica	54,7%

Tabla 9: Desglose Verdaderos Positivos ADN Polimerasa

Al igual que en caso anterior se apreció una disminución de mutaciones sinónimas idénticas entre las comparaciones con los genes alta y bajamente expresado.

III Discusión y Conclusiones

3.1 Discusión

La estrategia más común para la optimización de codones es reemplazar codones raros con codones más frecuentes, igualando así el CUB del organismo hospedero. La optimización tradicional de codones no siempre conduce a una mejora de la expresión en comparación a una secuencia tipo silvestre sin modificaciones. De hecho en una prueba realizada a 44 genes sintéticos manufacturados por Blue Heron Biotechnology, el 32% de los genes sintéticos "optimizados" se expresó en niveles más bajos que los de tipo salvaje (Blue Heron Biotechnology, 2014).

Lo anterior, recalca la importancia de una validación empírica para las herramientas que sugieren mutaciones. La validación de la herramienta diseñada y descrita en el presente trabajo se realizó mediante la comparación de resultados con los obtenidos de trabajos experimentales exhaustivos y detallados

En el caso de las mutaciones en el gen eGFP, que resultó ser altamente expresado en *Saccharomyces cerevisiae* (eGFP HT), el alto porcentaje de verdaderos positivos (95,2%) contrasta con el alto porcentaje de falsos negativos (49,0%). Al comparar las mutaciones en el gen eGFP, que resultaron en una baja expresión en *Saccharomyces cerevisiae* (eGFP C2), se apreció una disminución de 20 puntos porcentuales en los verdaderos positivos y el consecuente aumento del 20% en los falsos positivos.

Las diferencias en los resultados pueden deberse a que el enfoque de optimización de codones realizado en el presente trabajo descuida las condiciones de las células, y consideran toda la información del genoma de la proteína codificada como idéntica. Como resultado, este enfoque puede equivocarse al capturar los matices importantes de un CUB eficaz, que puede ser esencial para garantizar la expresión de un gen heterólogo.

En consideración a lo anterior, se ha demostrado que un CUB generado sólo por genes expresados bajo una condición dada puede permitir una mejor optimización de codones en *Saccharomyces cerevisiae,* en comparación a un CUB generado usando el uso de codones tabulado por GenBank (Lanza et al, 2014). Este enfoque se definió como: ‘condición especifica de optimización de codones’. Para esto, las condiciones bajo la cual el gen heterólogo de interés se expresará deben ser identificadas y los datos de expresión a escala genómica para el hospedero se deben obtener bajo esta condición. Se identifica los genes que, diferencialmente, estén sobre-regulados o altamente expresados bajo la condición deseada. De este conjunto de genes y sus correspondientes secuencias de ADN, se establece la frecuencia y probabilidad de los diferentes codones, permitiendo la determinación de un uso de codones (Lanza et al, 2014).

Para el caso de las mutaciones en el gen de ADN polimerasa, que resultó ser altamente expresado en *Escherichia coli*, se observó un alto porcentaje de verdaderos positivos (98,2%), también contrastado por un alto porcentaje de falsos negativos (93,5%). Al comparar con las mutaciones que resultaron en una baja expresión, también se observó

una disminución aproximada de 20 puntos porcentuales de verdaderos positivos y el consecuente aumento de cerca de un 20% de los falsos positivos. Contrariamente al caso anterior donde los verdaderos negativos y falsos negativos no tuvieron grandes variaciones entre los genes alta y bajamente expresados. El presente caso evidenció una disminución considerable para las secuencias débilmente expresadas (54,8%) de los falsos negativos. Esto se debió principalmente a la diferencia del número de mutaciones entre ambos casos. En P19 se mutó el 94,4% del total de codones del gen, mientras que en P15 se mutó el 61,1% del total de codones del gen.

Las diferencias en los resultado pueden deberse a los diferentes enfoques de optimización. La optimización del gen de la ADN polimerasa se centró en la identificación de las características de la secuencia que afectan a la expresión de la proteína. Se sintetizó y expresó 21 variantes del gen de ADN polimerasa en *Escherichia coli* y se encontró que la cantidad de proteína producida en *Escherichia coli* es, fuertemente, dependiente de los codones utilizados para codificar un subconjunto de aminoácidos. Los codones favorables eran, predominantemente, aquellos leídos por los ARNt que son más altamente cargados durante periodos de privación de aminoácidos, y no los codones que son más abundantes en las proteínas altamente expresadas en *Escherichia coli*.

Por consiguiente, si la carga de aminoácidos de ARNt se convierte en limitante, sólo los ARNt que puedan mantener una carga durante periodos de privación, pueden soportar altos niveles de traducción. Por lo tanto, el sesgo de codones óptimos para un gen, probablemente, depende tanto de mantener altos niveles de ARNt cargados como de minimizar los niveles de ARNt no cargados, que pueden inhibir la traducción y/o causar una respuesta metabólica perjudicial.

Al analizar los verdaderos positivos se pudo apreciar una disminución del 20% de las mutaciones de codones idénticas entre las comparaciones realizadas con los genes de eGFP, alta y bajamente expresados, y una disminución del 10% entre las comparaciones realizadas a los genes de ADN polimerasa. A modo de clarificar, si se toma un codón particular que es mutado y que al compararlo con el mismo codón en secuencia del gen altamente expresado son idénticos, y que posteriormente, al compararlo con el mismo codón pero en el gen bajamente expresado, estos difieren ya sea porque no se mutó el codón o porque se mutó a un codón sinónimo diferente. Esto puede significar que la mutación de ese particular codón resulte ser beneficioso para la optimización de la expresión.

Finalmente, las diferencias entre el número de mutaciones y los codones sinónimos elegidos para las mutaciones, pueden deberse a las diferentes bases de datos sobre las que se trabajó y a que el presente trabajo se enfocó a mutar para adecuar el gen a un uso de codones y abundancia de ARNt, a diferencia de los trabajos experimentales donde se mutó para adoptar un uso de codones especifico. Los trabajos experimentales mutan entre el 50% y el 90% del total de codones de una secuencia codificante, mientras que la herramienta bioinformática del presente trabajo muta alrededor del 15% del total de codones de una secuencia codificante.

3.2 Conclusiones

La producción de proteínas como agentes terapéuticos, reactivos de investigación y herramientas moleculares depende con frecuencia de la expresión en hospederos heterólogos. El análisis sistemático de los parámetros de diseño de genes permitió identificar el uso de codones y la abundancia de ARNt como factores determinantes para el diseño de genes heterólogos y la optimización de la expresión de proteínas recombinantes.

Se diseñó e implementó una herramienta de análisis y mutación sinónima de ADN en el software de programación MATLAB ®. Se ideó y conformó las bases de datos para el uso de codones y abundancia de ARNt de: *Escherichia coli*, *Bacillus subtilis*, *Saccharomyces cerevisiae*, *Cricetulus griseus*, Insectos dipteros, *Homo sapiens*, plantas briófitas, plantas angiospermas dicotiledóneas y plantas angiospermas monocotiledóneas.

Parte del diseño fue la creación de una interfaz amigable; clara en los conceptos aplicados y en las operación ofrecidas. Se diseñó un sistema de identificación de genes con un posterior análisis cuantitativo de su composición y cualitativo, en cuanto a su semejanza a las bases de datos. Adicionalmente, se diseñó una gráfica comparativa donde el usuario puede visualizar la totalidad de la extensión de la secuencia de ADN y comparar los datos propios de la secuencia con los datos provenientes de las bases de datos, que representan al organismo hospedero.

Se diseñó un algoritmo que reacondiciona el gen o secuencia al uso de codones y a la abundancia de ARNt definidos automática o manualmente por el usuario. Este acondicionamiento propone mutaciones sinónimas para la optimización de la expresión del gen en el organismo hospedero. Adicionalmente, se realizó y ofrece información al respecto de la presencia de una rampa al inicio de los genes, otorgando información sobre la presencia y abundancia de codones raros a lo largo del gen.

Si bien el uso de codones es un fuerte determinante del nivel de expresión general que se puede obtener de un gen, este nivel puede ser reducido si elementos nocivos están presentes en la secuencia, como por ejemplo los que forman estructuras secundarias en el ARNm que pueden interferir con la iniciación de la traducción.

Debido a lo anterior, se diseñó un análisis y la opción de visualizar gráficamente el plegamiento del ARN. El usuario tiene la posibilidad de modificar la secuencia para favorecer la disminución de la estabilidad de los plegamientos. Esto da como resultado final una lista de mutaciones, en consecuencia a las opciones elegidas por el usuario.

La comparación de resultados con trabajos experimentales abrió posibilidades para un futuro mejoramiento de diseño y conformación de las bases de datos. Entre los puntos destacables está la posibilidad de que las variaciones en la expresión de proteínas recombinantes, estén influidas por el agotamiento de los ARNt cargados o por la inducción de una respuesta metabólica desde el organismo hospedero. Esto reacondicionaría el uso de codones a los ARNt resistentes a periodos de privación de aminoácidos. Otro punto destacable es la cuantificación y análisis de las variaciones del

uso de codones bajo las condiciones específicas sobre las que se desea operar para la producción de la proteína recombinante. Ambos casos privilegian el uso de codones para la optimización de la expresión de genes recombinantes. El uso de codones para el organismo hospedero es acotado a condiciones específicas. Si bien, ambos trabajos presentan atractivos métodos de optimización estos no han sido comprobados en todos los dominios de la vida, lo que limita su inclusión en un trabajo de optimización de más amplio espectro.

Finalmente se puede decir, que se logró la implementación de una herramienta bioinformática de interfaz amigable y manipulable por el usuario que analiza numérica y gráficamente secuencias codificantes, y que además, sugiere mutaciones para la optimización de la expresión bajo el concepto de adaptar un gen a un organismo hospedero. A modo de complementar, se incluyó una revisión numérica y gráfica del plegamiento del ARNm, que amplía la capacidad de análisis y las opciones de mutación, siempre con el objetivo de lograr una alta eficiencia y fidelidad en la expresión de proteínas recombinantes.

IV Bibliografía

Adzhubei, A.A., Adzhubei, I.A., Krasheninnikov, I.A., Neidle, S. 1996. Non-random usage of 'degenerate' codons is related to protein three-dimensional structure. FEBS Lett 399, 78-82.

Akashi, H. 2003. Translational selection and yeast proteome evolution. Genetics 164, 1291-1303.

Akashi, H. y Eyre-Walker, A. 1998. Translational selection and molecular evolution. Current opinion in genetics & development 8, 688-693.

Alexeyev, M.F., Winkler, H.H. 1999. Gene synthesis bacterial expression and purification of the Rickettsia prowazekii ATP/ADP translocase. Biochim. Biophys. Acta 1419, 299-306.

Allawi, H.T., Santa Lucia, J.J. 1997. Thermodynamics and NMR of Internal G·T Mismatches in DNA. Biochemistry 36, 10581-10594.

Andersson, S.G.E. y Kurland, C.G. 1990. Codon preferences in free-living microorganisms. Microbiol Rev 54, 198-210.

Angov, E., Hillier, C.J., Kincaid, R.L., Lyon, J.A. 2008. Heterologous Protein Expression Is Enhanced by Harmonizing the Codon Usage Frequencies of the Target Gene with those of the Expression Host. PLoS One 3, e2189.

Bennetzen, N.L. y Hall, B.D. 1982. Codon selection in yeast. J.Biol. Chem. 257, 3026-3031.

Blanco, L., Salas, M. 1984. Characterization and purification of a phage phi 29-encoded DNA polymerase required for the initiation of replication. Proc Natl Acad Sci USA 81, 5325-5329.

Blue Heron Biotechnology. The Gene Synthesis Company. Expression Optimization Service Survey Results. [en línea] http://www.blueheronbio.com/assets/documents/BlueHeronBioExpressionSurvey.pdf [consulta: 3 julio 2014].

Cannarozzi, G., Schraudolph, N.N., Faty, M., von Rohr, P., Friberg, M.T., et al. 2010. A role for codon order in translation dynamics. Cell 141, 355-367.

Dekel, E. y Alon, U. 2005. Optimality and evolutionary tuning of the expression level of protein. Nature 436, 588-592.

Deng, T. 1997. Bacterial expression and purification of biologically active mouse c-Fos proteins by selective codon optimization. FEBS Lett. 409, 269-272.

Dong, H., Nilsson, L., Kurland, C.G. 1995. Gratuitous overexpression of genes in *Escherichia coli* leads to growth inhibition and ribosome destruction. J Bacteriol 177, 1497-1504.

Dos Reis, M., Savva, R., Wernisch, L. 2004. Solving the riddle of codon usage preferences: a test for translational selection. Nucleic Acids Res. 32, 5036-5044.

Drummond, D. y Wilke, C. 2008. Mistranslation-induced protein misfolding as a dominant constraint on coding-sequence evolution. Cell 134, 341-352.

Feng, L., Chan, W.W., Roderick, S.L., Cohen, D.E. 2000. High-level expression and mutagenesis of recombinant human phosphatidylcholine transfer protein using a synthetic gene: evidence for a C-terminal membrane binding domain. Biochemistry 39, 15399-15409.

Frelin, L., Ahlen, G., Alheim, M., Weiland, O., Barnfield, C., Liljestrom, P., Sallberg, M. 2004. Codon optimization and mRNA amplification effectively enhances the immunogenicity of the hepatitis C virus nonstructural 3/4A gene. Gene Ther. 11, 522-533.

Fuglsang, A. 2003. Codon optimizer: a freeware tool for codon optimization. Protein Expression and Purification 31, 247-249.

Genomic tRNA Database. tRNAscan-SE analysis of complete genomes. 1997. [en línea] http://lowelab.ucsc.edu/GtRNAdb/ [consulta: 3 julio 2014].

Gingold, H. y Pilpel, Y. 2011. Determinants of translation efficiency and accuracy. Mol Syst Biol 7, 481.

Haas, E. S., Banta, A. B., Harris, J. K., Pace, N. R., Brown, J. W. 1996. Structure and evolution of ribonuclease P RNA in Gram-positive bacteria. Nucleic Acids Res. 24, 4775-4782.

Hershberg, R., Petrov, D.A. 2009. General rules for optimal codon choice. PLoS Genet 5, e1000556.

Ikemura, T. 1981. Correlation between the abundance of *Escherichia coli* transfer RNAs and the occurrence of the respective codons in its protein genes: a proposal for a synonymous codon choice that is optimal for the E. coli translational system. J Mol Biol 151, 389-409.

Ikemura. 1982. Correlation between the abundance of yeast transfer RNAs and the occurrence of the respective codons in protein genes. J. Mol. Biol. 158, 573-597.

Ikemura, T. 1985. Codon usage and tRNA content in unicellular and multicellular organisms. Mol Biol Evol 2, 13-34.

Ingolia, N.T., Ghaemmaghami, S., Newman, J.R., Weissman, J.S. 2009. Genome-wide analysis in vivo of translation with nucleotide resolution using ribosome profiling. Science 324, 218-223.

Kazusa DNA Research Institute. Codon Usage Database. 2007. [en línea] http://www.kazusa.or.jp/codon/ [consulta: 3 julio 2014].

Kimchi-Sarfaty, C. et al. 2007. A 'silent' polymorphism in the MDR1 gene changes substrate specificity. Science 315, 525-528.

Komar, A., Lesnik, T., Reiss, C. 1999. Synonymous codon substitutions affect ribosome traffic and protein folding during in vitro translation. FEBS Lett 462, 387-391.

Kotula, L., Curtis, P.J. 1991. Evaluation of foreign gene codon optimization in yeast: expression of a mouse IG kappa chain. Biotechnology (NY) 9, 1386-1389.

Kudla, G. et al. 2009. Coding-sequence determinants of gene expression in *Escherichia coli*. Science 324, 255-258.

Kurland, C. y Gallant, J. 1996. Errors of heterologous protein expression. Curr Opin Biotechnol 7, 489-493.

Lanza, A.M., Curran, K.A., Rey, L.G., Alper, H.S. 2014. A condition-specific codon optimization approach for improved heterologous gene expression in *Saccharomyces cerevisiae*. BMC Syst. Biol. 8, 33.

Li, G.W., Oh, E., Weissman, J.S. 2012. The anti-Shine-Dalgarno sequence drives translational pausing and codon choice in bacteria. Nature 484, 538-541.

Li, X., Hirano, R., Tagami, H., Aiba, H. 2006. Protein tagging at rare codons is caused by tmRNA action at the 3' end of nonstop mRNA generated in response to ribosome stalling. RNA 12, 248-255.

Li, Z., Bo, X. 2014. GeneGA. Design gene based on both mRNA secondary structure and codon usage bias using Genetic algorithm. [en línea] Bioconductor: Open source software for bioinformatics. <http://www.bioconductor.org/> [consulta: 06 septiembre 2014].

Lindsley, D., Gallant, J., Guarneros, G. 2003. Ribosome bypassing elicited by tRNA depletion. Mol Microbiol 48, 1267-1274.

Lowe, T.M. y Eddy, S.R. 1997. tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence. Nucleic Acids Res. 25, 955-964.

Man, O. y Pilpel, Y. 2007. Differential translation efficiency of orthologous genes is involved in phenotypic divergence of yeast species. Nat. Genet. 39, 415-421.

Nakamura, Y., Gojobori, T., Ikemura, T. 1999. Codon usage tabulated from international DNA sequence databases: status for the year 2000. Nucleic Acids Res. 27, 292.

Nielsen, J. 2001. Metabolic Engineering. Applied Microbiology and Biotechnology 55, 263-283.

Parmley, J.L. y Hurst,L.D. 2007. How do synonymous mutations affect fitness? Bioessays, 29, 515-519.

Puigbo, P., Guzman, E., Romeu, A., Garcia-Vallve, S. 2007. OPTIMIZER: a web server for optimizing the codon usage of DNA sequences. Nucleic Acids Research. 35(2), W126-W131.

Qin, H., Wu, W.B., Comeron, J.M., Kreitman, M., Li, W.H. 2004. Intragenic spatial patterns of codon usage bias in prokaryotic and eukaryotic genomes. Genetics 168, 2245-2260.

Raab, D., Graf, M., Notka, F. 2009. The GeneOptimizer Algorithm: using a sliding window approach to cope with the vast sequence space in multiparameter DNA sequence optimization. Syst Synth Biol. 4, 215-225.

Raab, D., Graf, M., Notka, F., Schödl, T., Wagner, R. 2010. The GeneOptimizer Algorithm: using a sliding window approach to cope with the vast sequence space in multiparameter DNA sequence optimization. Syst Synth Biol. 4, 215-225.

Richardson, S.M., Wheelan, S.J., Yarrington, R.M., Boeke, J.D. 2006. GeneDesign: rapid, automated design of multikilobase synthetic genes. Genome Res. 16, 550-556.

Rocha, M., Maia, P., Mendes, R., Pinto, J.P., Ferreira, E.C., Nielsen, J., Patil, K.R., Rocha, I. 2008. Natural computation meta-heuristics for the in silico optimization of microbial strains. BMC Bioinformatics 9, 499.

Salis, H.M., Mirsky, E.A., Voigt, C.A. 2009. Automated design of synthetic ribosome binding sites to control protein expression. Nat. Biotechnol. 27, 946-950.

Santa Lucia, J.J. 1997. A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. Proc. Natl. Acad. Sci. USA 95, 1460-1465.

Shabalina, S.A., Ogurtsov, A.Y., Spiridonov, N.A. 2006. A periodic pattern of mRNA secondary structure created by the genetic code. Nucleic Acids Res., 34, 2428-2437.

Shabalina, S.A., Spiridonov, N.A., Kashina, A. 2012. Sounds of silence: synonymous nucleotides as a key to biological regulation and complexity. Nucleic Acids Res. 41, 2073-2094.

Shao, Z.Q., Zhang, Y.M., Feng, X.Y., Wang, B., Chen, J.Q. 2012. Synonymous Codon Ordering: A Subtle but Prevalent Strategy of Bacteria to Improve Translational Efficiency. PLoS One. 7, e33547

Sharp, P.M. y Li, W.H. 1987. The codon Adaptation Index-a measure of directional synonymous codon usage bias, and its potential applications. Nucleic Acids Res. 15, 1281-1295.

Sharp, P.M., Stenico, M., Peden, J.F., Lloyd, A.T. 1993. Codon usage: mutational bias, translational selection, or both?. Biochem Soc Trans 21, 835-841.

Sinclair, G., Choy, F.Y. 2002. Synonymous codon usage bias and the expression of human glucocerebrosidase in the methylotrophic yeast, Pichia pastoris. Protein Expr. Purif. 26, 96-105.

Stephanopoulos, G., Aristidou, A., Nielsen, J. 1998. Metabolic engineering, principles and methodologies. San Diego, California. Academic Press. 1, 1-20.

Tuller, T., Carmi, A., Vestsigian, K., Navon, S., Dorfan, Y., Zaborske, J., Pan, T., Dahan, O., Furman, I., Pilpel, Y. 2010. An Evolutionarily Conserved Mechanism for Controlling the Efficiency of Protein Translation. Cell 141, 344-354.

Tuller, T., Waldman, Y., Kupiec, M., Ruppin, E. 2010. Translation efficiency is determined by both codon bias and folding energy. Proc. Natl. Acad. Sci. USA 107, 3645-3650.

Villalobos, A., Ness, J.E., Gustafsson, C., Minshull, J., Govindarajan, S. 2006. Gene Designer: a synthetic biology tool for constructuring artificial DNA segments. BMC Bioinformatics 7, 285.

Welch, M., Govindarajan, S., Ness, J.E., Villalobos, A., Gurney, A., Minshull, J., Gustafsson, C. 2009. Design Parameters to Control Synthetic Gene Expression in *Escherichia coli*. PLoS One 4, e7002.

Wu, X., Jornvall, H., Berndt, K.D., Oppermann, U. 2004. Codon optimization reveals critical factors for high level expression of two rare codon genes in Escherichia coli: RNA stability and secondary structure but not tRNA abundance. Biochem. Biophys. Res. Commun. 313, 89-96.

Zhou, T., Weems, M., Wilke, C. 2009. Translationally optimal codons associate with structurally sensitive sites in proteins. Mol Biol Evol 26, 1571-1580.

Zur, H. y Tuller, T. 2012. Strong association between mRNA folding strength and protein abundance in S.cerevisiae. EMBO Rep. 13, 272-277.

V Anexos

5.1 Cálculo ΔG° de secuencias complementarias

$$5'\ C-G-T-T-G-A\ 3'$$
$$3'\ G-C-A-A-C-T\ 5'$$

$$\Delta G°_{37}(predicho) = \Delta G°\left(\frac{\mathrm{CG}}{\mathrm{GC}}\right) + \Delta G°\left(\frac{\mathrm{GT}}{\mathrm{CA}}\right) + \Delta G°\left(\frac{\mathrm{TT}}{\mathrm{AA}}\right) + \Delta G°\left(\frac{\mathrm{TG}}{\mathrm{AC}}\right) + \Delta G°\left(\frac{\mathrm{GA}}{\mathrm{CT}}\right) + \Delta G°(inicial) + \Delta G°(\mathrm{término})$$

Reemplazando por los valores ilustrados en la tabla 2.

$$\Delta G°_{37}(predicho) = -2.17 - 1.44 - 1.00 - 1.45 - 1.30 + 0.98 + 1.03$$
$$\Delta G°_{37}(predicho) = -5.35\ [\mathrm{kcal/mol}]$$
$$\Delta G°_{37}(observado) = -5.20\ [\mathrm{kcal/mol}]$$

5.2 Base de datos

5.2.1 Uso de Codones

	E.Coli	B.Subtilis	S.Cerevisiae	CHO	Insecta Diptera	Homo Sapiens	Bryophyta Bryopsida	Angiosperma Mono	Angiosperma Dicotiledónea
TTT	0,0223	0,0300	0,0261	0,0196	0,0136	0,0176	0,0172	0,0086	0,0252
TTC	0,0155	0,0143	0,0184	0,0220	0,0243	0,0203	0,0236	0,0259	0,0186
TTA	0,0140	0,0198	0,0262	0,0064	0,0043	0,0077	0,0079	0,0039	0,0135
TTG	0,0127	0,0158	0,0272	0,0141	0,0136	0,0129	0,0236	0,0094	0,0251
CTT	0,0121	0,0218	0,0123	0,0132	0,0084	0,0132	0,0166	0,0109	0,0221
CTC	0,0103	0,0107	0,0054	0,0184	0,0140	0,0196	0,0137	0,0216	0,0135
CTA	0,0040	0,0049	0,0134	0,0077	0,0076	0,0072	0,0077	0,0053	0,0112
CTG	0,0487	0,0230	0,0105	0,0388	0,0421	0,0396	0,0226	0,0309	0,0122
ATT	0,0292	0,0362	0,0301	0,0174	0,0148	0,0160	0,0208	0,0103	0,0261
ATC	0,0225	0,0272	0,0172	0,0248	0,0256	0,0208	0,0215	0,0391	0,0157
ATA	0,0061	0,0098	0,0178	0,0069	0,0088	0,0075	0,0083	0,0063	0,0137
ATG	0,0268	0,0263	0,0209	0,0230	0,0238	0,0220	0,0278	0,0213	0,0261
GTT	0,0184	0,0186	0,0221	0,0116	0,0108	0,0110	0,0180	0,0153	0,0233
GTC	0,0141	0,0173	0,0118	0,0157	0,0147	0,0145	0,0132	0,0429	0,0109
GTA	0,0113	0,0131	0,0118	0,0078	0,0073	0,0071	0,0096	0,0045	0,0104
GTG	0,0254	0,0173	0,0108	0,0301	0,0296	0,0281	0,0313	0,0309	0,0196
TCT	0,0088	0,0127	0,0235	0,0160	0,0057	0,0152	0,0150	0,0080	0,0211
TCC	0,0086	0,0083	0,0142	0,0165	0,0162	0,0177	0,0118	0,0350	0,0126
TCA	0,0089	0,0146	0,0187	0,0103	0,0067	0,0122	0,0116	0,0076	0,0187
TCG	0,0085	0,0065	0,0086	0,0035	0,0189	0,0044	0,0127	0,0117	0,0056
AGT	0,0097	0,0068	0,0142	0,0114	0,0096	0,0121	0,0110	0,0100	0,0141

AGC	0,0157	0,0144	0,0098	0,0164	0,0197	0,0195	0,0139	0,0109	0,0103
CCT	0,0081	0,0106	0,0135	0,0167	0,0058	0,0175	0,0159	0,0134	0,0187
CCC	0,0058	0,0035	0,0068	0,0170	0,0143	0,0198	0,0118	0,0180	0,0091
CCA	0,0084	0,0071	0,0183	0,0156	0,0116	0,0169	0,0125	0,0096	0,0183
CCG	0,0220	0,0163	0,0053	0,0043	0,0212	0,0069	0,0103	0,0149	0,0052
ACT	0,0094	0,0087	0,0203	0,0142	0,0078	0,0131	0,0153	0,0071	0,0177
ACC	0,0219	0,0090	0,0127	0,0203	0,0201	0,0189	0,0120	0,0200	0,0110
ACA	0,0090	0,0216	0,0178	0,0158	0,0095	0,0151	0,0123	0,0076	0,0155
ACG	0,0142	0,0149	0,0080	0,0045	0,0196	0,0061	0,0122	0,0120	0,0043
GCT	0,0154	0,0186	0,0212	0,0224	0,0132	0,0185	0,0241	0,0191	0,0231
GCC	0,0257	0,0165	0,0126	0,0259	0,0303	0,0277	0,0175	0,0296	0,0134
GCA	0,0215	0,0211	0,0162	0,0163	0,0133	0,0158	0,0201	0,0169	0,0198
GCG	0,0315	0,0198	0,0062	0,0050	0,0197	0,0074	0,0172	0,0149	0,0051
TAT	0,0168	0,0233	0,0188	0,0131	0,0097	0,0122	0,0105	0,0065	0,0179
TAC	0,0123	0,0126	0,0148	0,0164	0,0220	0,0153	0,0172	0,0120	0,0105
CAT	0,0137	0,0157	0,0136	0,0102	0,0103	0,0109	0,0111	0,0069	0,0179
CAC	0,0097	0,0075	0,0078	0,0129	0,0167	0,0151	0,0116	0,0097	0,0092
CAA	0,0141	0,0204	0,0273	0,0103	0,0136	0,0123	0,0167	0,0093	0,0222
CAG	0,0295	0,0185	0,0121	0,0334	0,0346	0,0342	0,0220	0,0163	0,0159
AAT	0,0202	0,0229	0,0357	0,0174	0,0176	0,0170	0,0178	0,0095	0,0261
AAC	0,0216	0,0178	0,0248	0,0212	0,0283	0,0191	0,0204	0,0187	0,0146
AAA	0,0336	0,0484	0,0419	0,0246	0,0158	0,0244	0,0189	0,0101	0,0292
AAG	0,0117	0,0208	0,0308	0,0384	0,0367	0,0319	0,0347	0,0582	0,0336
GAT	0,0333	0,0332	0,0376	0,0246	0,0261	0,0218	0,0278	0,0204	0,0349
GAC	0,0192	0,0190	0,0202	0,0281	0,0263	0,0251	0,0250	0,0442	0,0161
GAA	0,0391	0,0481	0,0456	0,0284	0,0215	0,0290	0,0241	0,0137	0,0324
GAG	0,0189	0,0226	0,0192	0,0411	0,0392	0,0396	0,0375	0,0454	0,0324
TGT	0,0055	0,0036	0,0081	0,0091	0,0072	0,0106	0,0070	0,0040	0,0108
TGC	0,0066	0,0043	0,0048	0,0104	0,0137	0,0126	0,0098	0,0080	0,0082
TGG	0,0147	0,0108	0,0104	0,0131	0,0109	0,0132	0,0138	0,0135	0,0156
CGT	0,0203	0,0072	0,0064	0,0056	0,0092	0,0045	0,0080	0,0137	0,0060
CGC	0,0207	0,0082	0,0026	0,0093	0,0187	0,0104	0,0085	0,0092	0,0047
CGA	0,0042	0,0043	0,0030	0,0072	0,0082	0,0062	0,0097	0,0078	0,0066
CGG	0,0065	0,0070	0,0017	0,0102	0,0120	0,0114	0,0095	0,0066	0,0052
AGA	0,0035	0,0105	0,0213	0,0101	0,0047	0,0122	0,0093	0,0060	0,0186
AGG	0,0023	0,0041	0,0092	0,0102	0,0050	0,0120	0,0109	0,0097	0,0160
GGT	0,0236	0,0130	0,0239	0,0128	0,0150	0,0108	0,0189	0,0097	0,0168
GGC	0,0274	0,0233	0,0098	0,0213	0,0255	0,0222	0,0185	0,0380	0,0109
GGA	0,0098	0,0218	0,0109	0,0158	0,0154	0,0165	0,0230	0,0093	0,0201
GGG	0,0114	0,0112	0,0060	0,0134	0,0073	0,0165	0,0160	0,0245	0,0147
TAA	0,0019	0,0019	0,0011	0,0006	0,0010	0,0010	0,0005	0,0003	0,0006
TAG	0,0002	0,0005	0,0005	0,0006	0,0007	0,0008	0,0007	0,0052	0,0006
TGA	0,0012	0,0008	0,0007	0,0012	0,0006	0,0016	0,0007	0,0007	0,0009

5.2.2 Abundancia de ARNt

	E.Coli	B.Subtilis	S.Cerevisiae	CHO	Insecta Diptera	Homo Sapiens	Bryophyta Bryopsida	Angiosperma Mono	Angiosperma Dicotiledónea
TTT	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0024	0,0003	0,0000
TTC	0,0220	0,0349	0,0383	0,0161	0,0243	0,0233	0,0333	0,0277	0,0344
TTA	0,0110	0,0349	0,0244	0,0080	0,0091	0,0136	0,0048	0,0058	0,0253
TTG	0,0110	0,0116	0,0348	0,0080	0,0103	0,0136	0,0143	0,0182	0,0199
CTT	0,0000	0,0000	0,0000	0,0100	0,0144	0,0233	0,0143	0,0240	0,0181
CTC	0,0110	0,0116	0,0035	0,0000	0,0000	0,0000	0,0024	0,0000	0,0000
CTA	0,0092	0,0233	0,0105	0,0080	0,0070	0,0058	0,0119	0,0128	0,0163
CTG	0,0398	0,0116	0,0000	0,0201	0,0255	0,0194	0,0119	0,0142	0,0127
ATT	0,0000	0,0000	0,0453	0,0261	0,0295	0,0272	0,0286	0,0428	0,0217
ATC	0,0263	0,0349	0,0035	0,0000	0,0000	0,0106	0,0000	0,0010	0,0000
ATA	0,0000	0,0000	0,0070	0,0100	0,0058	0,0097	0,0095	0,0064	0,0163
ATG	0,1060	0,0698	0,0383	0,0321	0,0418	0,0388	0,0571	0,0709	0,0470
GTT	0,0000	0,0000	0,0488	0,0261	0,0475	0,0213	0,0310	0,0232	0,0271
GTC	0,0220	0,0116	0,0000	0,0000	0,0000	0,0000	0,0000	0,0057	0,0000
GTA	0,0551	0,0465	0,0105	0,0100	0,0070	0,0097	0,0048	0,0060	0,0090
GTG	0,0000	0,0000	0,0070	0,0181	0,0467	0,0310	0,0262	0,0160	0,0199
TCT	0,0000	0,0000	0,0383	0,0121	0,0206	0,0213	0,0310	0,0157	0,0217
TCC	0,0220	0,0116	0,0000	0,0000	0,0000	0,0000	0,0000	0,0099	0,0018
TCA	0,0110	0,0233	0,0139	0,0060	0,0058	0,0097	0,0119	0,0149	0,0235
TCG	0,0110	0,0000	0,0035	0,0060	0,0164	0,0078	0,0167	0,0078	0,0054
AGT	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
AGC	0,0110	0,0233	0,0070	0,0181	0,0173	0,0155	0,0214	0,0205	0,0145
CCT	0,0000	0,0000	0,0070	0,0141	0,0202	0,0194	0,0310	0,0201	0,0217
CCC	0,0110	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
CCA	0,0124	0,0349	0,0348	0,0121	0,0180	0,0136	0,0191	0,0240	0,0271
CCG	0,0110	0,0000	0,0000	0,0060	0,0241	0,0078	0,0095	0,0107	0,0072
ACT	0,0000	0,0000	0,0383	0,0161	0,0247	0,0194	0,0238	0,0171	0,0181
ACC	0,0204	0,0116	0,0000	0,0020	0,0000	0,0000	0,0000	0,0061	0,0036
ACA	0,0140	0,0465	0,0174	0,0080	0,0125	0,0116	0,0119	0,0142	0,0163
ACG	0,0143	0,0000	0,0035	0,0080	0,0111	0,0116	0,0143	0,0083	0,0054
GCT	0,0000	0,0000	0,0383	0,0442	0,0406	0,0563	0,0452	0,0390	0,0199
GCC	0,0206	0,0116	0,0000	0,0020	0,0000	0,0000	0,0000	0,0000	0,0000
GCA	0,0294	0,0581	0,0209	0,0301	0,0070	0,0175	0,0214	0,0181	0,0850
GCG	0,0000	0,0000	0,0000	0,0121	0,0123	0,0097	0,0191	0,0183	0,0090
TAT	0,0000	0,0000	0,0000	0,0000	0,0000	0,0019	0,0000	0,0011	0,0018
TAC	0,0331	0,0233	0,0279	0,0221	0,0416	0,0272	0,0191	0,0227	0,0326
CAT	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0024	0,0018

CAC	0,0110	0,0233	0,0279	0,0281	0,0337	0,0213	0,0214	0,0265	0,0235
CAA	0,0185	0,0465	0,0314	0,0100	0,0115	0,0213	0,0214	0,0201	0,0145
CAG	0,0220	0,0000	0,0035	0,0221	0,0267	0,0398	0,0238	0,0169	0,0199
AAT	0,0000	0,0000	0,0000	0,0000	0,0000	0,0029	0,0024	0,0006	0,0018
AAC	0,0441	0,0465	0,0383	0,0341	0,0334	0,0621	0,0333	0,0306	0,0380
AAA	0,0648	0,0465	0,0279	0,0301	0,0197	0,0320	0,0191	0,0341	0,0199
AAG	0,0000	0,0000	0,0488	0,1727	0,0423	0,0330	0,0381	0,0552	0,0271
GAT	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0014	0,0036
GAC	0,0331	0,0465	0,0558	0,0301	0,0476	0,0369	0,0262	0,0384	0,0416
GAA	0,0509	0,0698	0,0523	0,0141	0,0233	0,0252	0,0167	0,0212	0,0235
GAG	0,0000	0,0000	0,0070	0,0402	0,0499	0,0252	0,0333	0,0315	0,0344
TGT	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0003	0,0000
TGC	0,0110	0,0116	0,0139	0,0823	0,0177	0,0582	0,0167	0,0217	0,0199
TGG	0,0110	0,0116	0,0209	0,0121	0,0206	0,0175	0,0214	0,0242	0,0181
CGT	0,0441	0,0465	0,0244	0,0121	0,0288	0,0136	0,0191	0,0266	0,0199
CGC	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0024	0,0005	0,0000
CGA	0,0102	0,0000	0,0000	0,0121	0,0264	0,0116	0,0167	0,0069	0,0090
CGG	0,0110	0,0116	0,0035	0,0060	0,0000	0,0087	0,0119	0,0089	0,0072
AGA	0,0422	0,0116	0,0418	0,0100	0,0074	0,0116	0,0143	0,0111	0,0109
AGG	0,0110	0,0116	0,0035	0,0121	0,0074	0,0097	0,0238	0,0136	0,0145
GGT	0,0000	0,0000	0,0000	0,0020	0,0000	0,0000	0,0000	0,0000	0,0000
GGC	0,0441	0,0465	0,0558	0,0281	0,0415	0,0291	0,0357	0,0371	0,0307
GGA	0,0140	0,0349	0,0105	0,0121	0,0197	0,0175	0,0214	0,0136	0,0217
GGG	0,0110	0,0000	0,0070	0,0141	0,0000	0,0136	0,0262	0,0113	0,0109
TAA	0,0000	0,0000	0,0000	0,0000	0,0000	0,0039	0,0000	0,0000	0,0036
TAG	0,0000	0,0000	0,0000	0,0000	0,0000	0,0019	0,0048	0,0017	0,0000
TGA	0,0110	0,0000	0,0035	0,0040	0,0017	0,0058	0,0024	0,0006	0,0018

5.3 Código

5.3.1 Código Central

```
function DSAM
ciclo = 1;
while ciclo
    [condx,codigo] = Enter_code;                % Primera ventana
    if ~isempty(codigo) && condx(1) == 1        % Condicion para seguir a 2
etapa
        ciclo2 = 1;
        while ciclo2
            codegen = rcode(codigo);                % Rescata el codigo del
archivo txt
            if condx(2) == 1
                [filcodegen] = filcode(condx(2),codegen);   % Filtra el codigo
codon inicio/termino
            elseif condx(2) == 0
                [filcodegen] = filcode(condx(2),codegen);   % condx(2) indica
si se filtra o no el codigo
            end
            if ~isempty(filcodegen)
                [numcodon,ordencod,cugen,ocodon] = codonusage(filcodegen);
                totalcod = sum(cugen);
                cond = 0;
                [indcu,cuinfo] = cu_check(cond,cugen);
                [indt,tinfo] = trna_check(cond,cugen);
                pcodon=cugen/totalcod;
                [condy,dbcond] = Det_code(totalcod,numcodon,ordencod,[indcu
indt]);  % Segunda ventana
                if condy == 1       % Condicion para seguir 3 etapa
                    ciclo3 = 1;
                    while ciclo3
                        if isequal(dbcond,[indcu,indt])
                            [codones] = valcodones(filcodegen,ocodon,pcodon);
                            [codonescu] =
valcodones(filcodegen,ocodon,cuinfo);
                            [codonest] = valcodones(filcodegen,ocodon,tinfo);
                            [condz] =
Comp_code(codones,codonescu,codonest,filcodegen,dbcond);
                        else
                            [indcu,cuinfo] = cu_check(dbcond(1),cugen);
                            [indt,tinfo] = trna_check(dbcond(2),cugen);
                            [codones] = valcodones(filcodegen,ocodon,pcodon);
                            [codonescu] =
valcodones(filcodegen,ocodon,cuinfo);
                            [codonest] = valcodones(filcodegen,ocodon,tinfo);
                            [condz] =
Comp_code(codones,codonescu,codonest,filcodegen,dbcond);
                        end
                        if condz == 1   % Condicion para seguir 4 etapa
                            [mjcodones] = mjvector(tinfo,ocodon);
                            if condx(2) == 1
                                [condw] =
Mut_gen(codones,codonescu,codonest,filcodegen,mjcodones);
                            elseif condx(2) == 0
                                [condw] =
Mut_gen2(codones,codonescu,codonest,filcodegen,mjcodones);
```

```
                            end
                            if condw == 1           % Condicion para seguir 5
etapa
                            elseif condw == 2       % Condicion para ir atras
                                ciclo3 = 1;
                            elseif condw == 0       % Condicion para salir
                                ciclo3 = 0;
                                ciclo2 = 0;
                                ciclo = 0;
                            end
                        elseif condz == 2   % Condicion para ir atras
                            ciclo3 = 0;
                            ciclo2 = 1;
                        elseif condz == 0   % Condicion para salir
                            ciclo3 = 0;
                            ciclo2 = 0;
                            ciclo = 0;
                        end
                    end
                elseif condy == 2   % Condicion para ir atras
                    ciclo2 = 0;
                    ciclo = 1;
                elseif condy == 0   % Condicion para Salir
                    ciclo2 = 0;
                    ciclo = 0;
                end
            end
        end
    elseif condx(1) == 0
        ciclo = 0;
    end
end
```

5.3.2 Código Ventanas

```
function [condx,code] = Enter_code(varargin)
if nargin == 0
    fig = openfig(mfilename,'reuse'
    set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));
    set(fig,'Name','Codon Analysis and Suggested Mutations');
    handles = guihandles(fig);
    handles.codegen = [];
    handles.cond = 0;
    handles.cbox = 1;
    guidata(fig, handles);
    set(handles.checkbox1,'Value',handles.cbox);
    set(handles.checkbox1,'String','Detection of the start and stop codon if
they exist');
    set(handles.text1,'String','File MUST be in FASTA Format.');
    set(handles.text2,'String','(Uncheck to read all the code without
distinction)');
    uiwait(fig);
    if ishandle(fig)
        handles = guidata(fig);
        code = handles.codegen;
        condx = [handles.cond,handles.cbox];
        delete(fig);
```

```
    else
        code = [];
        condx = 0;
    end
elseif ischar(varargin{1})
    try
        [varargout{1:nargout}] = feval(varargin{:});
    catch
        disp(lasterr);
    end
end

function edit1_CreateFcn(h, eventdata, handles)
if ispc && isequal(get(h,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(h,'BackgroundColor','white');
end

function varargout = edit1_Callback(h, eventdata, handles, varargin)
handles.codegen = get(handles.edit1,'String');
guidata(h,handles);

function varargout = pushbutton1_Callback(h, eventdata, handles, varargin)
if (~isempty(handles.codegen))
    handles.cond = 1;
    guidata(h,handles);
    uiresume(handles.figure1);
end

function varargout = pushbutton2_Callback(h, eventdata, handles, varargin)
handles.cdo = 0;
guidata(h,handles);
uiresume(handles.figure1);

function varargout = figure1_CloseRequestFcn(h, eventdata, handles, varargin)
guidata(h,handles);
uiresume(handles.figure1);

function varargout = checkbox1_Callback(h, eventdata, handles, varargin)
value = get(handles.checkbox1,'Value');
handles.cbox=value;
guidata(h,handles);

function [condy,bdatos] = Det_code(varargin)
if nargin == 0 | isnumeric(varargin{1})
    fig = openfig(mfilename,'reuse');
    set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));
    set(fig,'Name','Codon Analysis');
    handles = guihandles(fig);
    handles.totalcod = varargin{1};
    handles.numcodon = varargin{2};
    handles.ocodon = varargin{3};
    handles.indcu = varargin{4}(1);
    handles.indt = varargin{4}(2);
    handles.pcodon = handles.numcodon/handles.totalcod;
    handles.cond = 0;
```

```
    handles.sname={'escherichia coli','bacillus subtilis','saccharomyces
cerevisiae','chinese hamster ovary','insecta diptera',...
        'homo sapiens','bryophyta bryopsida','angiosperma monocotiledónea',
'angiosperma dicotiledónea'};
    bdatos=[handles.indcu,handles.indt];
    guidata(fig, handles);
cnames = {'%','n°codon'};
set(handles.uitable1,'Data',[handles.pcodon.*100,handles.numcodon],'ColumnName
',cnames,'RowName',handles.ocodon);
set(handles.popupmenu1,'Style','popup','String',handles.sname);
set(handles.popupmenu2,'Style','popup','String',handles.sname);
set(handles.popupmenu1,'Value',handles.indcu);
set(handles.popupmenu2,'Value',handles.indt);
set(handles.text1,'String','CU database:');
set(handles.text2,'String',handles.sname{handles.indcu});
set(handles.text3,'String','tRNA database:');
set(handles.text4,'String',handles.sname{handles.indt});
set(handles.text5,'String','Codon Usage and Species
Determination','FontSize',14);
set(handles.text6,'String','Graphical Analysis');
set(handles.text7,'String','Codon Usage Table:');
set(handles.text8,'String','Automatic Database Detection');
set(handles.text9,'String','Modify Automatic DB Detection');
set(handles.text10,'String','Database (DB):');
   uiwait(fig);
    if ishandle(fig)
        handles = guidata(fig);
        bdatos=[handles.indcu,handles.indt];
        condy = handles.cond;
        delete(fig);
    else
        condy = 0;
        bdatos=[0 0];
    end
elseif ischar(varargin{1})
    try
        [varargout{1:nargout}] = feval(varargin{:});
    catch
        disp(lasterr);
    end
end

function varargout = pushbutton1_Callback(h, eventdata, handles, varargin)
handles.cond = 1;
guidata(h,handles);
uiresume(handles.figure1);

function varargout = pushbutton2_Callback(h, eventdata, handles, varargin)
handles.cond = 2;
guidata(h,handles);
uiresume(handles.figure1);

function varargout = pushbutton3_Callback(h, eventdata, handles, varargin)
handles.cond = 0;
guidata(h,handles);
uiresume(handles.figure1);

function varargout = figure1_CloseRequestFcn(h, eventdata, handles, varargin)
```

```
guidata(h,handles);
uiresume(handles.figure1);

function varargout = popupmenu1_Callback(h, eventdata, handles, varargin)
value = get(handles.popupmenu1,'Value');
set(handles.text2,'String',handles.sname{value});
handles.indcu=value;
guidata(h,handles);

function popupmenu1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function varargout = popupmenu2_Callback(h, eventdata, handles, varargin)
value = get(handles.popupmenu2,'Value');
set(handles.text4,'String',handles.sname{value});
handles.indt=value;
guidata(h,handles);

function popupmenu2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function [condz] = Comp_code(varargin)
if nargin == 0 | isnumeric(varargin{1})
    fig = openfig(mfilename,'reuse');
    set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));
    set(fig,'Name','Graphical Analysis');
    handles = guihandles(fig);
    handles.codones = varargin{1};
    handles.codonescu = varargin{2};
    handles.codonest = varargin{3};
    handles.filcodegen = varargin{4};
    lcode=length(varargin{4});
    handles.dbcond = varargin{5};
    handles.num1 = 1;
    handles.num2 = floor(lcode/3);
    set(handles.edit1,'String','1');
    set(handles.edit2,'String',num2str(handles.num2));
    handles.cond = 0;
    guidata(fig, handles);
    sname={'escherichia coli','bacillus subtilis','saccharomyces
cerevisiae','chinese hamster ovary','insecta diptera',...
        'homo sapiens','bryophyta bryopsida','angiosperma monocotiledónea',
'angiosperma dicotiledónea'};
    set(handles.text1,'String','Graphical Analysis of the Code');
    set(handles.text2,'String','Codons');
    set(handles.text9,'String','Choose an interval of 20 or less codons to get
additional info');
    set(handles.text15,'String','0.0% to 0.5%');
    set(handles.text14,'String','0.5% to 1.0%');
    set(handles.text13,'String','1.0% to 1.5%');
    set(handles.text12,'String','1.5% to 2.0%');
    set(handles.text11,'String','2.0% to 4.0%');
```

```
    set(handles.text10,'String','greater than 4.0%');
    set(handles.text16,'String','% relating to each code and DB');
    set(handles.text17,'String','CU   Database: ');
    set(handles.text18,'String',sname(handles.dbcond(1)));
    set(handles.text19,'String','tRNA Database: ');
    set(handles.text20,'String',sname(handles.dbcond(2)));
    bargraph2(handles.codones,handles.codonescu,handles.codonest,...
       handles.filcodegen,handles.num1,handles.num2,...
       handles.figure1,handles.axes1);
    uiwait(fig);
    if ishandle(fig)
        handles = guidata(fig);
        condz = handles.cond;
        delete(fig);
    else
        condz = 0;
    end


elseif ischar(varargin{1})
    try
        [varargout{1:nargout}] = feval(varargin{:});
    catch
        disp(lasterr);
    end
end

function edit1_CreateFcn(h, eventdata, handles)
if ispc && isequal(get(h,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(h,'BackgroundColor','white');
end

function varargout = edit1_Callback(h, eventdata, handles, varargin)
handles.num1 = str2num(get(handles.edit1,'String'));
if handles.num1 < 1 || handles.num1 > floor(length(handles.filcodegen)/3)
    handles.num1 = 1;
end
guidata(h,handles);

function edit2_CreateFcn(h, eventdata, handles)
if ispc && isequal(get(h,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(h,'BackgroundColor','white');
end

function varargout = edit2_Callback(h, eventdata, handles, varargin)
handles.num2 = str2num(get(handles.edit2,'String'));
if handles.num2 > floor(length(handles.filcodegen)/3) || handles.num2 < 1
    handles.num2 = floor(length(handles.filcodegen)/3);
end
guidata(h,handles);

function varargout = pushbutton1_Callback(h, eventdata, handles, varargin)
if handles.num1 < handles.num2
    bargraph2(handles.codones,handles.codonescu,handles.codonest,...
        handles.filcodegen,handles.num1,handles.num2,...
```

```
        handles.figure1,handles.axes1);
else
    bargraph2(handles.codones,handles.codonescu,handles.codonest,...
        handles.filcodegen,1,floor(length(handles.filcodegen)/3),...
        handles.figure1,handles.axes1);
end

function varargout = pushbutton2_Callback(h, eventdata, handles, varargin)
handles.cond = 2;
guidata(h,handles);
uiresume(handles.figure1);

function varargout = pushbutton3_Callback(h, eventdata, handles, varargin)
handles.cond = 0;
guidata(h,handles);
uiresume(handles.figure1);

function varargout = pushbutton4_Callback(h, eventdata, handles, varargin)
handles.cond = 1;
guidata(h,handles);
uiresume(handles.figure1);

function varargout = figure1_CloseRequestFcn(h, eventdata, handles, varargin)
guidata(h,handles);
uiresume(handles.figure1);

function [condw] = Mut_gen(varargin)
if nargin == 0 | isnumeric(varargin{1})
    fig = openfig(mfilename,'reuse');
    set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));
    set(fig,'Name','Suggested Mutation');
    handles = guihandles(fig);
    handles.codones = varargin{1};
    handles.codonescu = varargin{2};
    handles.codonest = varargin{3};
    handles.filcodegen = varargin{4};
    handles.mcusage = varargin{5};
    handles.ramp = 50;
    handles.cond = 0;
    guidata(fig, handles);
    set(handles.edit1,'String',num2str(handles.ramp));
    [porcs] = counteramp(handles.ramp,handles.codones);
    set(handles.text3,'String',{['Rare Codons in the Ramp
',num2str(porcs(1),'%.2f'),'%'],...
        ['Rare Codons out of the Ramp  ',num2str(porcs(3),'%.2f'),'%']});
    set(handles.text4,'String',{['Rare codons in the ramp zone
',num2str(porcs(2),'%.2f'),'%'],...
        ['Rare codons out of the ramp zone  ',num2str(porcs(4),'%.2f'),'%']});
    set(handles.text5,'String',['Total Rare Codons
',num2str(porcs(5),'%.2f'),'%']);
    set(handles.text6,'String','Codons in the ramp N°Codon-AA-Codon');
    set(handles.text7,'String','Codons out of the ramp N°Codon-AA-Codon');
    set(handles.text8,'String','Suggested Mutation');
    set(handles.text9,'String','Suggested Mutation');
    set(handles.text10,'String','Ramp Size: Choose a value from 20 to 50');
    set(handles.text11,'String','Folding Analysis on:');
    set(handles.text12,'String','Mutated Sequence: First Stage (DBs)');
    set(handles.text1,'String','First Stage: tRNA and Codon Usage database');
```

```
    set(handles.text2,'String','Second Stage: Synonymous Codon Pairing and
Major Codon Usage');
    set(handles.text13,'String','Additional Option: Analyze Sequence without
Mutations');
    set(handles.text14,'String','Mutation Stages: check to add filters');
    set(handles.text15,'String','Synonymous Codon Mutation');
    codigomod = codemod(handles.filcodegen);
    [codigo] =
mutationer(codigomod,handles.codonest,handles.codonescu,handles.ramp);
    [codigocs] = cspair(codigo,codigomod);
    [indmut,aamut,codonesmut,codonesori] = defmut(codigomod,codigo);
    [codigomc] = mjcodon(codigomod,codigocs,handles.mcusage);
    handles.codigo = codigo;
    handles.codigocs = codigocs;
    handles.codigomc = codigomc;
    handles.foldcodigo = codigo;
    guidata(fig, handles);
    indsep=0;
    for i = 1:length(indmut)
        if indmut(i)>handles.ramp
            indsep = i-1;
            break
        end
    end
    codigo1=[];
    codigocs1=[];
    codigomc1=[];
    for i=1:length(codigo)
        codigo1=strcat(codigo1,codigo(i,1:3));
        codigocs1=strcat(codigocs1,codigocs(i,1:3));
        codigomc1=strcat(codigomc1,codigomc(i,1:3));
    end
feandgc=[cgcontent(codigomod),cgcontent(codigo),cgcontent(codigocs),cgcontent(
codigomc);...
foldenergy(handles.filcodegen),foldenergy(codigo1),foldenergy(codigocs1),folde
nergy(codigomc1)];
    cnames1={'Sequence','Mutated Sequence(DBs)','Mutated
Sequence(SCP)','Mutated Sequence(MCU)'};
    cnames2={'%GC','dG'};
set(handles.listbox1,'String',[num2str(indmut(1:indsep)'),aamut(1:indsep,1:3),
upper(codonesori(1:indsep,1:3))],...
       'FontSize',12,'FontAngle','italic');
set(handles.listbox2,'String',[num2str(indmut((indsep+1):end)'),aamut((indsep+
1):end,1:3),upper(codonesori((indsep+1):end,1:3))],...
       'FontSize',12,'FontAngle','italic','HorizontalAlignment','right');
    set(handles.listbox3,'String',upper(codonesmut(1:indsep,1:3)),...
       'FontSize',12,'FontAngle','italic');
set(handles.listbox4,'String',upper(codonesmut((indsep+1):end,1:3)),...
       'FontSize',12,'FontAngle','italic','HorizontalAlignment','right');
    set(handles.listbox3,'Enable','inactive');
    set(handles.listbox4,'Enable','inactive');
    set(handles.checkbox1,'Value',1,'Enable','inactive');
set(handles.uitable1,'Data',feandgc,'ColumnName',cnames1,'RowName',cnames2);
    uiwait(fig);
    if ishandle(fig)
        handles = guidata(fig);
        condw = handles.cond;
        delete(fig);
    else
```

```
        condw = 0;
    end
elseif ischar(varargin{1
    try
        [varargout{1:nargout}] = feval(varargin{:});
    catch
        disp(lasterr);
    end
end

function varargout = listbox1_Callback(h, eventdata, handles,varargin)
value = get(handles.listbox1,'Value');
set(handles.listbox3,'Value',value);
guidata(h,handles);

function listbox1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function varargout = listbox2_Callback(h, eventdata, handles,varargin)
value = get(handles.listbox2,'Value');
set(handles.listbox4,'Value',value);
guidata(h,handles);

function listbox2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function varargout = pushbutton1_Callback(h, eventdata, handles, varargin)
[porcs] = counteramp(handles.ramp,handles.codones);
set(handles.text3,'String',{['Rare Codons in the Ramp
',num2str(porcs(1),'%.2f'),'%'],...
    ['Rare Codons out of the Ramp  ',num2str(porcs(3),'%.2f'),'%']});
set(handles.text4,'String',{['Rare codons in the ramp zone
',num2str(porcs(2),'%.2f'),'%'],...
    ['Rare codons out of the ramp zone  ',num2str(porcs(4),'%.2f'),'%']});
set(handles.text5,'String',['Total Rare Codons
',num2str(porcs(5),'%.2f'),'%']);
set(handles.checkbox2,'Value',0);
set(handles.checkbox3,'Value',0);
set(handles.listbox1,'Value',1);
set(handles.listbox2,'Value',1);
set(handles.listbox3,'Value',1);
set(handles.listbox4,'Value',1);
codigomod = codemod(handles.filcodegen);
[codigo] =
mutationer(codigomod,handles.codonest,handles.codonescu,handles.ramp);
[codigocs] = cspair(codigo,codigomod);
[indmut,aamut,codonesmut,codonesori] = defmut(codigomod,codigo);
[codigomc] = mjcodon(codigomod,codigocs,handles.mcusage);
handles.codigo = codigo;
handles.codigocs = codigocs;
handles.codigomc = codigomc;
handles.foldcodigo = codigo;
```

```
indsep=0;
for i = 1:length(indmut)
    if indmut(i)>handles.ramp
        indsep = i-1;
        break
    end
end
codigo1=[];
codigocs1=[];
codigomc1=[];
for i=1:length(codigo)
    codigo1=strcat(codigo1,codigo(i,1:3));
    codigocs1=strcat(codigocs1,codigocs(i,1:3));
    codigomc1=strcat(codigomc1,codigomc(i,1:3));
end
feandgc=[cgcontent(codigomod),cgcontent(codigo),cgcontent(codigocs),cgcontent(
codigomc);...
foldenergy(handles.filcodegen),foldenergy(codigo1),foldenergy(codigocs1),folde
nergy(codigomc1)];
set(handles.uitable1,'Data',feandgc);
set(handles.listbox1,'String',[num2str(indmut(1:indsep)'),aamut(1:indsep,1:3),
upper(codonesori(1:indsep,1:3))],...
    'FontSize',12,'FontAngle','italic');
set(handles.listbox2,'String',[num2str(indmut((indsep+1):end)'),aamut((indsep+
1):end,1:3),upper(codonesori((indsep+1):end,1:3))],...
    'FontSize',12,'FontAngle','italic','HorizontalAlignment','right');
set(handles.listbox3,'String',upper(codonesmut(1:indsep,1:3)),...
    'FontSize',12,'FontAngle','italic');
set(handles.listbox4,'String',upper(codonesmut((indsep+1):end,1:3)),...
    'FontSize',12,'FontAngle','italic','HorizontalAlignment','right');
set(handles.text12,'String','Mutated Sequence: First Stage (DBs)');
guidata(h, handles);

function varargout = pushbutton2_Callback(h, eventdata, handles, varargin)
Fold_code(handles.codonest,handles.codonescu,handles.foldcodigo,handles.filcod
egen);

function varargout = edit1_Callback(h, eventdata, handles, varargin)
handles.ramp = str2num(get(handles.edit1,'String'));
if handles.ramp < 20
    handles.ramp = 20;
elseif handles.ramp > 50
    handles.ramp = 50;
end
guidata(h,handles);

function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function varargout = figure1_CloseRequestFcn(h, eventdata, handles, varargin)
guidata(h,handles);
uiresume(handles.figure1);

function varargout = listbox3_Callback(h, eventdata, handles,varargin)
```

```
function listbox3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function varargout = listbox4_Callback(h, eventdata, handles,varargin)

function listbox4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function varargout = pushbutton3_Callback(h, eventdata, handles, varargin)
handles.cond = 2;
guidata(h,handles);
uiresume(handles.figure1);

function varargout = pushbutton4_Callback(h, eventdata, handles, varargin)
handles.cond = 0;
guidata(h,handles);
uiresume(handles.figure1);

function  varargout = checkbox1_Callback(h, eventdata, handles, varargin)

function  varargout = checkbox2_Callback(h, eventdata, handles, varargin)
value = get(handles.checkbox2,'Value');
codigomod = codemod(handles.filcodegen);
if value == 0 && get(handles.checkbox4,'Value')~=1
    set(handles.checkbox3,'Value',value);
    [indmut,aamut,codonesmut,codonesori] = defmut(codigomod,handles.codigo);
    indsep=0;
    for i = 1:length(indmut)
        if indmut(i)>handles.ramp
            indsep = i-1;
            break
        end
    end
set(handles.listbox1,'String',[num2str(indmut(1:indsep)'),aamut(1:indsep,1:3),
upper(codonesori(1:indsep,1:3))],...
        'FontSize',12,'FontAngle','italic');
set(handles.listbox2,'String',[num2str(indmut((indsep+1):end)'),aamut((indsep+
1):end,1:3),upper(codonesori((indsep+1):end,1:3))],...
        'FontSize',12,'FontAngle','italic','HorizontalAlignment','right');
set(handles.listbox3,'String',upper(codonesmut(1:indsep,1:3)),...
        'FontSize',12,'FontAngle','italic');
set(handles.listbox4,'String',upper(codonesmut((indsep+1):end,1:3)),...
       'FontSize',12,'FontAngle','italic','HorizontalAlignment','right');
    handles.foldcodigo = handles.codigo;
    set(handles.text12,'String','Mutated Sequence: First Stage (DBs)');
elseif value == 1 && get(handles.checkbox4,'Value')~=1
    [indmut,aamut,codonesmut,codonesori] = defmut(codigomod,handles.codigocs);
    indsep=0;
    for i = 1:length(indmut)
        if indmut(i)>handles.ramp
            indsep = i-1;
            break
```

```
        end
    end
    set(handles.listbox1,'Value',1);
    set(handles.listbox2,'Value',1);
    set(handles.listbox3,'Value',1);
    set(handles.listbox4,'Value',1);
set(handles.listbox1,'String',[num2str(indmut(1:indsep)'),aamut(1:indsep,1:3),
upper(codonesori(1:indsep,1:3))],...
        'FontSize',12,'FontAngle','italic');
set(handles.listbox2,'String',[num2str(indmut((indsep+1):end)'),aamut((indsep+
1):end,1:3),upper(codonesori((indsep+1):end,1:3))],...
'FontSize',12,'FontAngle','italic','HorizontalAlignment','right');
set(handles.listbox3,'String',upper(codonesmut(1:indsep,1:3)),...
        'FontSize',12,'FontAngle','italic');
set(handles.listbox4,'String',upper(codonesmut((indsep+1):end,1:3)),...
      'FontSize',12,'FontAngle','italic','HorizontalAlignment','right');
handles.foldcodigo = handles.codigocs;
set(handles.text12,'String','Mutated Sequence: Second Stage (SCP)');
end
guidata(h,handles);

function  varargout = checkbox3_Callback(h, eventdata, handles, varargin)
value = get(handles.checkbox3,'Value');
codigomod = codemod(handles.filcodegen);
if value == 1 && get(handles.checkbox4,'Value')~=1
    set(handles.checkbox2,'Value',value)
     [indmut,aamut,codonesmut,codonesori] =
defmut(codigomod,handles.codigomc);
    indsep=0;
    for i = 1:length(indmut)
        if indmut(i)>handles.ramp
            indsep = i-1;
            break
        end
    end
    set(handles.listbox1,'Value',1);
    set(handles.listbox2,'Value',1);
    set(handles.listbox3,'Value',1);
    set(handles.listbox4,'Value',1);
set(handles.listbox1,'String',[num2str(indmut(1:indsep)'),aamut(1:indsep,1:3),
upper(codonesori(1:indsep,1:3))],...
        'FontSize',12,'FontAngle','italic');
set(handles.listbox2,'String',[num2str(indmut((indsep+1):end)'),aamut((indsep+
1):end,1:3),upper(codonesori((indsep+1):end,1:3))],...
'FontSize',12,'FontAngle','italic','HorizontalAlignment','right');
set(handles.listbox3,'String',upper(codonesmut(1:indsep,1:3)),...
        'FontSize',12,'FontAngle','italic');
set(handles.listbox4,'String',upper(codonesmut((indsep+1):end,1:3)),...
       'FontSize',12,'FontAngle','italic','HorizontalAlignment','right');
    handles.foldcodigo = handles.codigomc;
    set(handles.text12,'String','Mutated Sequence: Second Stage (MCU)');
elseif value == 0 && get(handles.checkbox4,'Value')~=1
    [indmut,aamut,codonesmut,codonesori] = defmut(codigomod,handles.codigocs);
    indsep=0;
    for i = 1:length(indmut)
        if indmut(i)>handles.ramp
            indsep = i-1;
            break
        end
```

```
    end
    set(handles.listbox1,'Value',1);
    set(handles.listbox2,'Value',1);
    set(handles.listbox3,'Value',1);
    set(handles.listbox4,'Value',1);
set(handles.listbox1,'String',[num2str(indmut(1:indsep)'),aamut(1:indsep,1:3),
upper(codonesori(1:indsep,1:3))],...
        'FontSize',12,'FontAngle','italic');
set(handles.listbox2,'String',[num2str(indmut((indsep+1):end)'),aamut((indsep+
1):end,1:3),upper(codonesori((indsep+1):end,1:3))],...
'FontSize',12,'FontAngle','italic','HorizontalAlignment','right');
set(handles.listbox3,'String',upper(codonesmut(1:indsep,1:3)),...
        'FontSize',12,'FontAngle','italic');
set(handles.listbox4,'String',upper(codonesmut((indsep+1):end,1:3)),...
'FontSize',12,'FontAngle','italic','HorizontalAlignment','right');
    handles.foldcodigo = handles.codigocs;
    set(handles.text12,'String','Mutated Sequence: Second Stage (SCP)');
end
guidata(h,handles);

function varargout = checkbox4_Callback(h, eventdata, handles, varargin)
value = get(handles.checkbox4,'Value');
if value == 1
    set(handles.checkbox2,'Value',0,'Enable','inactive'
    set(handles.checkbox3,'Value',0,'Enable','inactive');
    codigomod = codemod(handles.filcodegen);
    handles.foldcodigo = codigomod;
    set(handles.text12,'String','Sequence without mutations');
elseif value == 0
    codigomod = codemod(handles.filcodegen);
    [indmut,aamut,codonesmut,codonesori] = defmut(codigomod,handles.codigo);
    indsep=0;
    for i = 1:length(indmut)
        if indmut(i)>handles.ramp
            indsep = i-1;
            break
        end
    end
set(handles.listbox1,'String',[num2str(indmut(1:indsep)'),aamut(1:indsep,1:3),
upper(codonesori(1:indsep,1:3))],...
        'FontSize',12,'FontAngle','italic');
set(handles.listbox2,'String',[num2str(indmut((indsep+1):end)'),aamut((indsep+
1):end,1:3),upper(codonesori((indsep+1):end,1:3))],...
'FontSize',12,'FontAngle','italic','HorizontalAlignment','right');
set(handles.listbox3,'String',upper(codonesmut(1:indsep,1:3)),...
        'FontSize',12,'FontAngle','italic');
set(handles.listbox4,'String',upper(codonesmut((indsep+1):end,1:3)),...
'FontSize',12,'FontAngle','italic','HorizontalAlignment','right');
    set(handles.checkbox2,'Enable','on');
    set(handles.checkbox3,'Enable','on');
    handles.foldcodigo = handles.codigo;
    set(handles.text12,'String','Mutated Sequence: First Stage (DBs)');
end
guidata(h,handles);

function [condw] = Mut_gen2(varargin)
if nargin == 0 | isnumeric(varargin{1})
    fig = openfig(mfilename,'reuse');
```

```
    set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));
    set(fig,'Name','Suggested Mutation');
    handles = guihandles(fig);
    handles.codones = varargin{1};
    handles.codonescu = varargin{2};
    handles.codonest = varargin{3};
    handles.filcodegen = varargin{4};
    handles.mcusage = varargin{5};
    handles.ramp = 0;
    handles.cond = 0;
    guidata(fig, handles);
    set(handles.text6,'String','N°Codon-AA-Codon');
    set(handles.text8,'String','Suggested Mutation');
    set(handles.text11,'String','Folding Analysis on:');
    set(handles.text12,'String','Mutated Sequence: First Stage (DBs)');
    set(handles.text1,'String','First Stage: tRNA and Codon Usage database');
    set(handles.text2,'String','Second Stage: Synonymous Codon Pairing and
Major Codon Usage');
    set(handles.text13,'String','Additional Option: Analyze Sequence without
Mutations');
    set(handles.text14,'String','Mutation Stages: check to add filters');
    set(handles.text15,'String','Synonymous Codon Mutation');
    codigomod = codemod(handles.filcodegen);
    [codigo] =
mutationer(codigomod,handles.codonest,handles.codonescu,handles.ramp);
    [codigocs] = cspair(codigo,codigomod);
    [indmut,aamut,codonesmut,codonesori] = defmut(codigomod,codigo);
    [codigomc] = mjcodon(codigomod,codigocs,handles.mcusage);
    handles.codigo = codigo;
    handles.codigocs = codigocs;
    handles.codigomc = codigomc;
    handles.foldcodigo = codigo;
    guidata(fig, handles);
    codigo1=[];
    codigocs1=[];
    codigomc1=[];
    for i=1:length(codigo)
        codigo1=strcat(codigo1,codigo(i,1:3));
        codigocs1=strcat(codigocs1,codigocs(i,1:3));
        codigomc1=strcat(codigomc1,codigomc(i,1:3));
    end
feandgc=[cgcontent(codigomod),cgcontent(codigo),cgcontent(codigocs),cgcontent(
codigomc);...
foldenergy(handles.filcodegen),foldenergy(codigo1),foldenergy(codigocs1),folde
nergy(codigomc1)];
    cnames1={'Sequence','Mutated Sequence(DBs)','Mutated
Sequence(SCP)','Mutated Sequence(MCU)'};
    cnames2={'%GC','dG'};
set(handles.listbox1,'String',[num2str(indmut'),aamut(1:end,1:3),upper(codones
ori(1:end,1:3))],...
        'FontSize',12,'FontAngle','italic');
    set(handles.listbox2,'String',upper(codonesmut(1:end,1:3)),...
        'FontSize',12,'FontAngle','italic');
    set(handles.listbox2,'Enable','inactive');
    set(handles.checkbox1,'Value',1,'Enable','inactive');
set(handles.uitable1,'Data',feandgc,'ColumnName',cnames1,'RowName',cnames2);
    uiwait(fig);
    if ishandle(fig)
        handles = guidata(fig);
```

```
        condw = handles.cond;
        delete(fig);
    else
        condw = 0;
    end
elseif ischar(varargin{1})
    try
        [varargout{1:nargout}] = feval(varargin{:});
    catch
        disp(lasterr);
    end
end

function varargout = listbox1_Callback(h, eventdata, handles,varargin)
value = get(handles.listbox1,'Value');
set(handles.listbox2,'Value',value);
guidata(h,handles);

function listbox1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function varargout = listbox2_Callback(h, eventdata, handles,varargin)

function listbox2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function varargout = pushbutton2_Callback(h, eventdata, handles, varargin)
Fold_code(handles.codonest,handles.codonescu,handles.foldcodigo,handles.filcod
egen);

function varargout = figure1_CloseRequestFcn(h, eventdata, handles, varargin)
guidata(h,handles);
uiresume(handles.figure1);

function varargout = pushbutton3_Callback(h, eventdata, handles, varargin)
handles.cond = 2;
guidata(h,handles);
uiresume(handles.figure1);

function varargout = pushbutton4_Callback(h, eventdata, handles, varargin)
handles.cond = 0;
guidata(h,handles);
uiresume(handles.figure1);

function  varargout = checkbox1_Callback(h, eventdata, handles, varargin)

function  varargout = checkbox2_Callback(h, eventdata, handles, varargin)
value = get(handles.checkbox2,'Value');
codigomod = codemod(handles.filcodegen);
if value == 0 && get(handles.checkbox4,'Value')~=1
    set(handles.checkbox3,'Value',value);
```

```
    [indmut,aamut,codonesmut,codonesori] = defmut(codigomod,handles.codigo);
set(handles.listbox1,'String',[num2str(indmut'),aamut(1:end,1:3),upper(codones
ori(1:end,1:3))],...
        'FontSize',12,'FontAngle','italic');
    set(handles.listbox2,'String',upper(codonesmut(1:end,1:3)),...
        'FontSize',12,'FontAngle','italic');
    handles.foldcodigo = handles.codigo;
    set(handles.text12,'String','Mutated Sequence: First Stage (DBs)');
elseif value == 1 && get(handles.checkbox4,'Value')~=1
    [indmut,aamut,codonesmut,codonesori] = defmut(codigomod,handles.codigocs);
    set(handles.listbox1,'Value',1);
    set(handles.listbox2,'Value',1);
set(handles.listbox1,'String',[num2str(indmut'),aamut(1:end,1:3),upper(codones
ori(1:end,1:3))],...
        'FontSize',12,'FontAngle','italic');
    set(handles.listbox2,'String',upper(codonesmut(1:end,1:3)),...
        'FontSize',12,'FontAngle','italic');
    handles.foldcodigo = handles.codigocs;
    set(handles.text12,'String','Mutated Sequence: Second Stage (SCP)');
end
guidata(h,handles);

function  varargout = checkbox3_Callback(h, eventdata, handles, varargin)
value = get(handles.checkbox3,'Value');
codigomod = codemod(handles.filcodegen);
if value == 1 && get(handles.checkbox4,'Value')~=1
    set(handles.checkbox2,'Value',value)
     [indmut,aamut,codonesmut,codonesori] =
defmut(codigomod,handles.codigomc);
    set(handles.listbox1,'Value',1);
    set(handles.listbox2,'Value',1);
set(handles.listbox1,'String',[num2str(indmut'),aamut(1:end,1:3),upper(codones
ori(1:end,1:3))],...
        'FontSize',12,'FontAngle','italic');
    set(handles.listbox2,'String',upper(codonesmut(1:end,1:3)),...
        'FontSize',12,'FontAngle','italic');
    handles.foldcodigo = handles.codigomc;
    set(handles.text12,'String','Mutated Sequence: Second Stage (MCU)');
elseif value == 0 && get(handles.checkbox4,'Value')~=1
    [indmut,aamut,codonesmut,codonesori] = defmut(codigomod,handles.codigocs);
    set(handles.listbox1,'Value',1);
    set(handles.listbox2,'Value',1);
set(handles.listbox1,'String',[num2str(indmut'),aamut(1:end,1:3),upper(codones
ori(1:end,1:3))],...
        'FontSize',12,'FontAngle','italic');
    set(handles.listbox2,'String',upper(codonesmut(1:end,1:3)),...
        'FontSize',12,'FontAngle','italic');
    handles.foldcodigo = handles.codigocs;
    set(handles.text12,'String','Mutated Sequence: Second Stage (SCP)');
end
guidata(h,handles);

function varargout = checkbox4_Callback(h, eventdata, handles, varargin)
value = get(handles.checkbox4,'Value');
if value == 1
    set(handles.checkbox2,'Value',0,'Enable','inactive');
    set(handles.checkbox3,'Value',0,'Enable','inactive');
    codigomod = codemod(handles.filcodegen);
```

```
    handles.foldcodigo = codigomod;
    set(handles.text12,'String','Sequence without mutations');
elseif value == 0
    set(handles.listbox1,'Value',1);
    set(handles.listbox2,'Value',1);
    codigomod = codemod(handles.filcodegen);
    [indmut,aamut,codonesmut,codonesori] = defmut(codigomod,handles.codigo);
set(handles.listbox1,'String',[num2str(indmut'),aamut(1:end,1:3),upper(codones
ori(1:end,1:3))],...
        'FontSize',12,'FontAngle','italic');
    set(handles.listbox2,'String',upper(codonesmut(1:end,1:3)),...
        'FontSize',12,'FontAngle','italic');
    set(handles.checkbox2,'Enable','on');
    set(handles.checkbox3,'Enable','on');
    handles.foldcodigo = handles.codigo;
    set(handles.text12,'String','Mutated Sequence: First Stage (DBs)');
end
guidata(h,handles);

function varargout = Fold_code(varargin)
if nargin == 0 || isnumeric(varargin{1})
    fig = openfig(mfilename,'reuse');
    set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));
    set(fig,'Name','Folding Analysis');
    handles = guihandles(fig);
    handles.codonest = varargin{1};
    handles.codonescu = varargin{2};
    handles.mutcode = varargin{3};
    handles.filcodegen = varargin{4};
    handles.rango = 5;
    guidata(fig, handles);
    foldcodigo=[];
    for i=1:length(handles.mutcode)
      foldcodigo=strcat(foldcodigo,handles.mutcode(i,1:3));
    end
    foldind=foldtest(foldcodigo);

    handles.foldcodigo = foldcodigo;
    handles.foldind = foldind;
    guidata(fig, handles);
    foldseq={};
    for j = 1:length(foldind(:,1))
foldseq(j,:)={upper(handles.foldcodigo(foldind(j,1):foldind(j,2))),strcat(num2
str(foldind(j,1)),'-',num2str(foldind(j,2))),...
upper(handles.foldcodigo(foldind(j,3):foldind(j,4))),strcat(num2str(foldind(j,
3)),'-',num2str(foldind(j,4)))...
foldind(j,5)};
    end
    cnames1 = {'Sequence','Range','Rev Comp Seq','Range','dG'};
    cnames2 = {'Location','Codon','Mut
Codon','Classification','N°Codon','AA'};
    set(handles.uitable1,'Data',foldseq,'ColumnName',cnames1);
    set(handles.uitable2,'ColumnName',cnames2);
    set(handles.uitable3,'ColumnName',cnames1);
    set(handles.togglebutton2,'Enable','inactive');
    set(handles.edit1,'String',num2str(handles.rango));
    set(handles.text1,'String','Sequence/Reverse Complementary Sequence
Mutation');
```

```
    set(handles.text2,'String','Select a Maximum for Nucleotide Folding');
    set(handles.text3,'String','(Process could take several minutes)');
    set(handles.text4,'String','Synonymous Mutation and Folding Recalculate');
    set(handles.text6,'String','Folding Structure');
    set(handles.text7,'String','Folding Structure');
    set(handles.text8,'String','Integrated List of Mutations');
    set(handles.text9,'String','(Process could take several minutes)');
    uiwait(fig);
    if ishandle(fig)
        delete(fig);
    end
elseif ischar(varargin{1})
    try
        [varargout{1:nargout}] = feval(varargin{:});
    catch
        disp(lasterr);
    end
end

function varargout = figure1_CloseRequestFcn(h, eventdata, handles, varargin)
guidata(h,handles);
uiresume(handles.figure1);

function varargout = edit1_Callback(h, eventdata, handles, varargin)
handles.rango = str2num(get(handles.edit1,'String'));
if handles.rango < 4
    handles.rango = 4;
    set(handles.edit1,'String',num2str(handles.rango));
elseif handles.rango >= length(handles.foldind(1,1):handles.foldind(1,2))
    handles.rango = length(handles.foldind(1,1):handles.foldind(1,2))-1;
    set(handles.edit1,'String',num2str(handles.rango));
end
guidata(h,handles);

function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function varargout = pushbutton1_Callback(h, eventdata, handles, varargin)
codxfil=[];
[codfoldmut,mutotal1] =
mutfoldcod(handles.mutcode,handles.codonest,handles.codonescu,...
    handles.foldind,handles.rango,codxfil,0);
codfoldmutfinal=codfoldmut;
mutotal2=[];
codigoe=[];
    for i=1:length(handles.mutcode)
        codigoe=strcat(codigoe,codfoldmut(i,1:3));
    end
foldinde=foldtest(codigoe);
if length(foldinde(1,1):foldinde(1,2))>handles.rango
    codxfil=mutotal1(:,1)';
    [codfoldmut,mutotal2] =
mutfoldcod(codfoldmut,handles.codonest,handles.codonescu,...
        foldinde,handles.rango,codxfil,0);
    codfoldmutfinal=codfoldmut;
```

```
end
mutotal=vertcat(mutotal1,mutotal2);
[indmut,aamut,codonesmut,codonesori,condmut] =
defmutfold(handles.mutcode,codfoldmutfinal,mutotal);
foldcodones={};
    for i = 1:length(indmut)
        classmut=[];
        if condmut(i)==1
            classmut='Convenient';
        elseif condmut(i)==2
            classmut='Not Convenient';
        else
            classmut='Perilous';
        end
        foldcodones(i,:)={strcat(num2str(indmut(i)*3-2),'-
',num2str(indmut(i)*3)),...
upper(codonesori(i,1:3)),upper(codonesmut(i,1:3)),classmut,indmut(i),aamut(i,1
:3)};
    end
set(handles.uitable2,'Data',foldcodones);
foldcodigo2=[];
    for i=1:length(handles.mutcode)
        foldcodigo2=strcat(foldcodigo2,codfoldmutfinal(i,1:3));
    end
foldind2=foldtest(foldcodigo2);
handles.foldcodigo2 = foldcodigo2;
handles.foldind2 = foldind2;
foldseq={};
    for j = 1:length(foldind2(:,1))
foldseq(j,:)={upper(handles.foldcodigo2(foldind2(j,1):foldind2(j,2))),strcat(n
um2str(foldind2(j,1)),'-',num2str(foldind2(j,2))),...
upper(handles.foldcodigo2(foldind2(j,3):foldind2(j,4))),strcat(num2str(foldind
2(j,3)),'-',num2str(foldind2(j,4)))...
            foldind2(j,5)};
    end
set(handles.uitable3,'Data',foldseq);
codigomod = codemod(handles.filcodegen);
[indmutx,aamutx,codonesmutx,codonesorix] = defmut(codigomod,codfoldmut);
set(handles.listbox1,'String',[num2str(indmutx'),aamutx(1:end,1:3),upper(codon
esmutx(1:end,1:3))],...
        'FontSize',12,'FontAngle','italic');
set(handles.togglebutton2,'Enable','on');
guidata(h,handles);

function varargout = togglebutton1_Callback(h, eventdata, handles, varargin)
value = get(handles.togglebutton1,'Value');
if value == 1
    set(handles.togglebutton1,'Enable','inactive');
    set(handles.togglebutton1,'String','Ploting');
    linea=pregraf(handles.foldind,length(handles.foldcodigo));
    rnaplot(linea,'seq',handles.foldcodigo,'format','dot');
    set(handles.togglebutton1,'String','Ploted');
end
guidata(h,handles);

function varargout = togglebutton2_Callback(h, eventdata, handles, varargin)
value = get(handles.togglebutton2,'Value');
if value == 1
```

```
    set(handles.togglebutton2,'Enable','inactive');
    set(handles.togglebutton2,'String','Ploting');
    linea=pregraf(handles.foldind2,length(handles.foldcodigo2));
    rnaplot(linea,'seq',handles.foldcodigo2,'format','dot');
    set(handles.togglebutton2,'String','Ploted');
end
guidata(h,handles);

function varargout = pushbutton2_Callback(h, eventdata, handles, varargin)
uiresume(handles.figure1);

function varargout = listbox1_Callback(h, eventdata, handles, varargin)

function listbox1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

5.3.3 Código Funciones

```
Function
bargraph2(codones,codonescu,codonest,filcodegen,num1,num2,fighandle,graphandle
)
codemo=codemod(filcodegen);
codeaa=codigoaa(codemo);
n = num2-num1+1;
x=[(num1-1):(num2-1);(num1-1):(num2-1);num1:num2;num1:num2];
y=zeros(4,n);
y(2:3,:)=ones(2,n);
yc=y+1.5;
yt=yc+1.5;
z=ones(4,n);
x1=[(num1-0.5):1:(num2-0.5)];
y1=ones(1,n);
y1=y1./2;
y1c=y1+1.5;
y1t=y1c+1.5;
cdata1=[];
cdata1(:,:,1)=zeros(1,n);
cdata1(:,:,2)=zeros(1,n);
cdata1(:,:,3)=zeros(1,n);
gen=codones;
 for i=(num1):(num2)
    if (gen(i)>=0) && (gen(i)<=0.005)
    cdata1(:,(i-num1+1),1:3)=[0.8 0 0];
    elseif (gen(i)>0.005) && (gen(i)<=0.01)
    cdata1(:,(i-num1+1),1:3)=[1 0 0];
    elseif (gen(i)>0.01) && (gen(i)<=0.015)
    cdata1(:,(i-num1+1),1:3)=[1 1 0];
    elseif (gen(i)>0.015) && (gen(i)<=0.02)
    cdata1(:,(i-num1+1),1:3)=[0 1 0];
    elseif (gen(i)>0.02) && (gen(i)<=0.04)
    cdata1(:,(i-num1+1),1:3)=[0 0.8 0];
    elseif (gen(i)>0.04) && (gen(i)<1)
    cdata1(:,(i-num1+1),1:3)=[0 0.6 0];
    end
 end
```

```
cdata2=[];
cdata2(:,:,1)=zeros(1,n);
cdata2(:,:,2)=zeros(1,n);
cdata2(:,:,3)=zeros(1,n);
genc=codonescu;
 for i=(num1):(num2)
    if (genc(i)>=0) && (genc(i)<=0.005)
    cdata2(:,(i-num1+1),1:3)=[0.8 0 0];
    elseif (genc(i)>0.005) && (genc(i)<=0.01)
    cdata2(:,(i-num1+1),1:3)=[1 0 0];
    elseif (genc(i)>0.01) && (genc(i)<=0.015)
    cdata2(:,(i-num1+1),1:3)=[1 1 0];
    elseif (genc(i)>0.015) && (genc(i)<=0.02)
    cdata2(:,(i-num1+1),1:3)=[0 1 0];
    elseif (genc(i)>0.02) && (genc(i)<=0.04)
    cdata2(:,(i-num1+1),1:3)=[0 0.8 0];
    elseif (genc(i)>0.04) && (genc(i)<1)
    cdata2(:,(i-num1+1),1:3)=[0 0.6 0];
    end
 end
cdata3=[];
cdata3(:,:,1)=zeros(1,n);
cdata3(:,:,2)=zeros(1,n);
cdata3(:,:,3)=zeros(1,n);
gent=codonest;
 for i=(num1):(num2)
    if (gent(i)>=0) && (gent(i)<=0.005)
    cdata3(:,(i-num1+1),1:3)=[0.8 0 0];
    elseif (gent(i)>0.005) && (gent(i)<=0.01)
    cdata3(:,(i-num1+1),1:3)=[1 0 0];
    elseif (gent(i)>0.01) && (gent(i)<=0.015)
    cdata3(:,(i-num1+1),1:3)=[1 1 0];
    elseif (gent(i)>0.015) && (gent(i)<=0.02)
    cdata3(:,(i-num1+1),1:3)=[0 1 0];
    elseif (gent(i)>0.02) && (gent(i)<=0.04)
    cdata3(:,(i-num1+1),1:3)=[0 0.8 0];
    elseif (gent(i)>0.04) && (gent(i)<1)
    cdata3(:,(i-num1+1),1:3)=[0 0.6 0];
    end
 end
set(0,'CurrentFigure',fighandle);
set(fighandle,'CurrentAxes',graphandle);
set(graphandle,'NextPlot','replace');
handle_text=findobj(graphandle,'Type','text');
delete(handle_text);
handle_patch=findobj(graphandle,'Type','patch');
delete(handle_patch);
pstar=patch(x,y,z,'Parent',graphandle);
set(graphandle,'NextPlot','add');
pcu=patch(x,yt,z,'Parent',graphandle);
pt=patch(x,yc,z,'Parent',graphandle);
set(graphandle,'YTickLabel',{' ','Code',' ','
','DB','tRNA','DB','Usage','Codon'})
set(pstar,'FaceColor','flat','CData',cdata1);
set(pstar,'EdgeColor','none');
set(pcu,'FaceColor','flat','CData',cdata2);
set(pcu,'EdgeColor','none');
set(pt,'FaceColor','flat','CData',cdata3);
set(pt,'EdgeColor','none');
```

```
set(graphandle,'XLim',[num1-1 num2]);
set(graphandle,'Layer','top');
if (num2-num1+1)<= 21
    for i=num1:num2
    str = {upper(codemo(i,(1:3))),codeaa(i),[num2str(gen(i)*100,'%.2f'),'%']};
    text(x1(i-num1+1),y1(i-
num1+1),str,'HorizontalAlignment','center','FontSize',8,...
        'BackgroundColor','w','EdgeColor','k','Parent',graphandle);
    strc = {upper(codemo(i,(1:3))),[num2str(genc(i)*100,'%.2f'),'%']};
    text(x1(i-num1+1),y1t(i-
num1+1),strc,'HorizontalAlignment','center','FontSize',8,...
        'BackgroundColor','w','EdgeColor','k','Parent',graphandle);
    strt = {upper(codemo(i,(1:3))),[num2str(gent(i)*100,'%.2f'),'%']};
    text(x1(i-num1+1),y1c(i-
num1+1),strt,'HorizontalAlignment','center','FontSize',8,...
        'BackgroundColor','w','EdgeColor','k','Parent',graphandle);
    end
    set(pstar,'EdgeColor','k');
    set(pcu,'EdgeColor','k');
    set(pt,'EdgeColor','k');
end
set(graphandle,'NextPlot','replace');

function [porc] = cgcontent(secuencia)
sec=[];
for i=1:length(secuencia)
    sec=strcat(sec,secuencia(i,1:3));
end
sec2=lower(sec);
ccont=find(sec2=='c');
gcont=find(sec2=='g');
cgcont=length(ccont)+length(gcont);
porc=cgcont*100/length(sec2);
end

function [codemo] = codemod( code )
i=1;
codemo=[];
while i <= length(code)-2
codemo=[codemo;code(i:i+2)];
i=i+3;
end
end

function [codaa] = codigoaa(codgene)
matriz =
['FTTT';'FTTC';'LTTA';'LTTG';'LCTT';'LCTC';'LCTA';'LCTG';'IATT';'IATC';'IATA';
'MATG';'VGTT';'VGTC';'VGTA';'VGTG';'STCT';'STCC';'STCA';...
'STCG';'SAGT';'SAGC';'PCCT';'PCCC';'PCCA';'PCCG';'TACT';'TACC';'TACA';'TACG';'
AGCT';'AGCC';'AGCA';'AGCG';'YTAT';'YTAC';'HCAT';'HCAC';...
'QCAA';'QCAG';'NAAT';'NAAC';'KAAA';'KAAG';'DGAT';'DGAC';'EGAA';'EGAG';'CTGT';'
CTGC';'WTGG';'RCGT';'RCGC';'RCGA';'RCGG';'RAGA';'RAGG';...
        'GGGT';'GGGC';'GGGA';'GGGG';'XTAA';'XTAG';'XTGA'];
codaa=[];
largo = length(codgene);

i = 1;
while i <= largo
```

```
j=1;
    while j <= 64
        if strcmpi(codgene(i,1:3),matriz(j,(2:4)))
        codaa=[codaa,matriz(j,1)];
        break
        end
    j=j+1;
    end
i=i+1;
end

function [numcodon,ordencod,ncodon,matriz] = codonusage(codgene)
matriz =
['FTTT';'FTTC';'LTTA';'LTTG';'LCTT';'LCTC';'LCTA';'LCTG';'IATT';'IATC';'IATA';
'MATG';'VGTT';'VGTC';'VGTA';'VGTG';'STCT';'STCC';'STCA';...
'STCG';'SAGT';'SAGC';'PCCT';'PCCC';'PCCA';'PCCG';'TACT';'TACC';'TACA';'TACG';'
AGCT';'AGCC';'AGCA';'AGCG';'YTAT';'YTAC';'HCAT';'HCAC';...
'QCAA';'QCAG';'NAAT';'NAAC';'KAAA';'KAAG';'DGAT';'DGAC';'EGAA';'EGAG';'CTGT';'
CTGC';'WTGG';'RCGT';'RCGC';'RCGA';'RCGG';'RAGA';'RAGG';...
        'GGGT';'GGGC';'GGGA';'GGGG';'XTAA';'XTAG';'XTGA'];
ncodon=zeros(64,1);
largo = length(codgene);
ordencod=[];
i = 1;
while i <= (largo-2)
j=1;
    while j <= 64
        if strcmpi(codgene(i:(i+2)),matriz(j,(2:4)))
        ncodon(j,1)=ncodon(j,1)+1;
        break
        end
    j=j+1;
    end
i=i+3;
end
[numcodon,ind]=sort(ncodon,'descend');
m=1;
while m <= 64
    ordencod=[ordencod;matriz(ind(m),2:4)];
    m=m+1;
end
end

function [porcentajes] = counteramp(numramp,codones)
ramp=0;
noramp=0;
porcentajes=[];
for i = 1:numramp
   if codones(i)<0.01
   ramp=ramp+1;
   end
end
for j = (numramp+1):length(codones)
   if codones(j)<0.01
   noramp=noramp+1;
   end
end
porcentajes=[roundn(ramp*100/(ramp+noramp),-2),roundn(ramp*100/numramp,-2),...
```

```
    roundn(noramp*100/(ramp+noramp),-2),roundn(noramp*100/(length(codones)-
numramp),-2),...
    roundn((ramp+noramp)*100/length(codones),-2)];
end

function [codigocs] = cspair(codigo,codigomod)
largo=length(codigo);
codaa = codigoaa(codigomod);
codigocs = codigomod;
aa=[];
for i = 1:largo
    if ~strcmpi(codigo(i,1:3),codigomod(i,1:3))
        aa=codaa(i);
        if i>=2 && ~isempty(find(codaa(1:(i-1))==aa,1,'last'))
            if strcmpi(codigocs(find(codaa(1:(i-
1))==aa,1,'last'),1:3),codigo(i,1:3))
                codigocs(i,1:3)=codigo(i,1:3);
            end
        else
            codigocs(i,1:3)=codigo(i,1:3);
        end
    end
end
end

function [ind,sequence] = cu_check(cond,cugen)
matriz=[0.02228 0.01554 0.01401 0.01269 0.01209 0.01034 0.00404 0.04873
0.02918 0.02248 0.00614 0.02677 0.01841 0.01413 0.01127 0.02541 0.0088
0.00857 0.00886 0.00849 0.00969 0.01568 0.00805 0.00576 0.00842 0.02199
0.00941 0.0219  0.00902 0.01423 0.01536 0.02569 0.02149 0.03152 0.01683
0.01234 0.01368 0.00973 0.01409 0.02953 0.02015 0.02162 0.03356 0.01174
0.03333 0.01918 0.03907 0.01887 0.00554 0.00663 0.01466 0.02029 0.02073
0.00422 0.00647 0.00352 0.00232 0.02355 0.02739 0.00975 0.01144 0.00189
0.00023 0.0012;...
        0.02998 0.01432 0.01983 0.01584 0.02179 0.01067 0.00487 0.023
0.03616 0.02718 0.00976 0.02627 0.01863 0.01725 0.01305 0.0173  0.01266 0.0083
0.01456 0.00646 0.00684 0.01443 0.0106  0.0035  0.00711 0.01633 0.00868 0.009
0.02163 0.01485 0.01858 0.01647 0.0211  0.01984 0.02326 0.01262 0.01574
0.00753 0.02038 0.0185  0.02293 0.01781 0.04838 0.02084 0.03324 0.01899
0.04809 0.0226  0.0036  0.00426 0.01075 0.00724 0.00824 0.00434 0.00695 0.0105
0.0041  0.01296 0.02326 0.02176 0.01115 0.00192 0.00047 0.00076;...
        0.02612 0.01844 0.02615 0.02717 0.01225 0.00544 0.01341 0.01048
0.03013 0.01717 0.01779 0.02094 0.02207 0.01178 0.01177 0.01076 0.0235
0.01422 0.01867 0.00856 0.01415 0.00975 0.01351 0.00678 0.01831 0.00529
0.02028 0.01273 0.01776 0.00796 0.02117 0.0126  0.01621 0.00618 0.01878
0.01478 0.01362 0.00777 0.02728 0.01211 0.03568 0.02482 0.04187 0.03082
0.03759 0.02021 0.0456  0.01924 0.0081  0.00476 0.01037 0.0064  0.0026
0.00299 0.00174 0.02128 0.00923 0.02389 0.00978 0.0109  0.00602 0.00106
0.00051 0.00068;...
        0.01957 0.02202 0.00637 0.01413 0.01318 0.01836 0.00765 0.03879
0.01741 0.0248  0.00686 0.02304 0.01159 0.01568 0.00783 0.03014 0.01596
0.01647 0.01027 0.00345 0.01144 0.01642 0.01669 0.01699 0.01555 0.00428
0.01415 0.02031 0.01575 0.00446 0.02235 0.02588 0.01626 0.00498 0.01314
0.01641 0.01018 0.0129  0.01034 0.03336 0.0174  0.02116 0.02463 0.0384
0.02463 0.02807 0.02837 0.04111 0.0091  0.01035 0.01311 0.00562 0.00931
0.00718 0.01015 0.01014 0.01023 0.01282 0.02129 0.0158  0.01344 0.00061
0.00055 0.00115;...
```

```
        0.01358 0.02429 0.00432 0.01356 0.00836 0.014   0.00758 0.04212
0.01481 0.02561 0.00876 0.02383 0.01084 0.0147  0.00734 0.02956 0.00572
0.01623 0.00672 0.01887 0.0096  0.01969 0.00579 0.01434 0.0116  0.0212
0.00783 0.02013 0.0095  0.0196  0.01322 0.03031 0.01332 0.01969 0.00969 0.022
0.01031 0.01668 0.01359 0.03462 0.01764 0.02829 0.01583 0.03672 0.02609
0.02626 0.02152 0.03924 0.00723 0.01373 0.01088 0.00915 0.01873 0.00816
0.01196 0.0047  0.00495 0.015   0.02547 0.01536 0.00733 0.001   0.00069
0.00058;...
        0.01757 0.02028 0.00767 0.01293 0.01319 0.01959 0.00715 0.03964 0.016
0.02082 0.00749 0.02204 0.01103 0.01446 0.00708 0.02812 0.01522 0.01768
0.01221 0.00441 0.01213 0.01946 0.01754 0.01979 0.01692 0.00692 0.01312
0.01889 0.01511 0.00605 0.01845 0.02773 0.01582 0.00737 0.01219 0.01531
0.01086 0.01509 0.01234 0.03423 0.01696 0.0191  0.02444 0.03186 0.02178 0.0251
0.02896 0.03959 0.01058 0.01262 0.01317 0.00454 0.01042 0.00617 0.01142
0.01217 0.01196 0.01075 0.02222 0.01647 0.01647 0.00099 0.00079 0.00156;...
        0.01716 0.02357 0.0079  0.02361 0.01661 0.0137  0.00772 0.02256
0.02077 0.02154 0.00825 0.02784 0.01797 0.01319 0.00957 0.03126 0.01495 0.0118
0.01157 0.0127  0.01095 0.01386 0.01592 0.01181 0.01246 0.01025 0.01533
0.01195 0.01227 0.0122  0.02407 0.01746 0.02006 0.01722 0.01047 0.01722
0.01113 0.01158 0.0167  0.02196 0.01775 0.02036 0.01892 0.03474 0.02777
0.02496 0.02411 0.03745 0.007   0.00984 0.01382 0.00798 0.00851 0.00972
0.00953 0.00933 0.01086 0.01885 0.01849 0.023   0.01604 0.00051 0.0007
0.00067;...
        0.00861 0.02591 0.00389 0.00935 0.01093 0.02162 0.00534 0.03087
0.01025 0.03911 0.00632 0.02128 0.01528 0.04285 0.00454 0.03093 0.00797
0.03502 0.00761 0.01166 0.00997 0.01085 0.01344 0.018   0.00964 0.01489
0.00706 0.01998 0.00755 0.01202 0.0191  0.02959 0.01687 0.01485 0.0065
0.01199 0.00694 0.00974 0.00928 0.01625 0.00949 0.01874 0.0101  0.05817
0.02043 0.0442  0.0137  0.04535 0.004   0.00804 0.01345 0.01367 0.00916
0.00783 0.0066  0.00598 0.00968 0.00966 0.03796 0.00927 0.02445 0.00027
0.00517 0.00073;...
        0.02522 0.01863 0.0135  0.02506 0.02211 0.01354 0.01115 0.01221 0.0261
0.01569 0.01371 0.02613 0.02326 0.01093 0.01041 0.01963 0.02114 0.01263
0.01869 0.00559 0.01408 0.01025 0.0187  0.0091  0.0183  0.00523 0.01772
0.01101 0.0155  0.00432 0.02311 0.01343 0.01984 0.00506 0.0179  0.01051
0.01788 0.00915 0.02216 0.01592 0.02609 0.01455 0.02923 0.03363 0.03491
0.01611 0.03236 0.03236 0.01083 0.00817 0.01561 0.00603 0.00465 0.00662
0.00518 0.01859 0.01601 0.01684 0.01094 0.02006 0.01465 0.00059 0.00059
0.0009];
ind=0;
if cond == 0
    j=1;
    totalvar=[];
    pcugen=cugen/sum(cugen);
    while j <= 9
        sumvar=0;
        for i=1:64
            sumvar=sumvar+((pcugen(i)-
(pcugen(i)+matriz(j,i))/2)^2+(matriz(j,i)-(pcugen(i)+matriz(j,i))/2)^2);
        end
        totalvar(j)=sumvar;
        minvar = min(totalvar);
        if minvar == totalvar(j)
            ind=j;
        end
    j=j+1;
    end
    sequence=matriz(ind,:);
else
```

```
    ind = cond;
    sequence=matriz(ind,:);
end
end

function [ncodon,matriz] = cusagemut(codgene)
matriz =
['FTTT';'FTTC';'LTTA';'LTTG';'LCTT';'LCTC';'LCTA';'LCTG';'IATT';'IATC';'IATA';
'MATG';'VGTT';'VGTC';'VGTA';'VGTG';'STCT';'STCC';'STCA';...

'STCG';'SAGT';'SAGC';'PCCT';'PCCC';'PCCA';'PCCG';'TACT';'TACC';'TACA';'TACG';'
AGCT';'AGCC';'AGCA';'AGCG';'YTAT';'YTAC';'HCAT';'HCAC';...

'QCAA';'QCAG';'NAAT';'NAAC';'KAAA';'KAAG';'DGAT';'DGAC';'EGAA';'EGAG';'CTGT';'
CTGC';'WTGG';'RCGT';'RCGC';'RCGA';'RCGG';'RAGA';'RAGG';...
        'GGGT';'GGGC';'GGGA';'GGGG';'XTAA';'XTAG';'XTGA'];
lmatriz=length(matriz);
ncodon=zeros(lmatriz,1);
largo = length(codgene);
i = 1;
while i <= largo
j=1;
    while j <= lmatriz
        if strcmpi(codgene(i,1:3),matriz(j,(2:4)))
        ncodon(j,1)=ncodon(j,1)+1;
        break
        end
        j=j+1;
    end
i=i+1;
end

function [ind,aamut,codonesmut,codonesori] = defmut(codori,codmut)
matriz =
['FTTT';'FTTC';'LTTA';'LTTG';'LCTT';'LCTC';'LCTA';'LCTG';'IATT';'IATC';'IATA';
'MATG';'VGTT';'VGTC';'VGTA';'VGTG';'STCT';'STCC';'STCA';...
'STCG';'SAGT';'SAGC';'PCCT';'PCCC';'PCCA';'PCCG';'TACT';'TACC';'TACA';'TACG';'
AGCT';'AGCC';'AGCA';'AGCG';'YTAT';'YTAC';'HCAT';'HCAC';...
'QCAA';'QCAG';'NAAT';'NAAC';'KAAA';'KAAG';'DGAT';'DGAC';'EGAA';'EGAG';'CTGT';'
CTGC';'WTGG';'RCGT';'RCGC';'RCGA';'RCGG';'RAGA';'RAGG';...
        'GGGT';'GGGC';'GGGA';'GGGG';'XTAA';'XTAG';'XTGA'];
ind=[];
codonesmut=[];
codonesori=[];
aamut=[];
aa=[];
i=1;
while i <= length(codori)
    if ~strcmpi(codori(i,1:3),codmut(i,1:3))
        ind=[ind,i];
        codonesmut=[codonesmut;codmut(i,1:3)];
        codonesori=[codonesori;codori(i,1:3)];
    end
    i=i+1;
end
for j = 1:length(ind)
    k=1;
    while k <= 64
```

```
        if strcmpi(codonesmut(j,1:3),matriz(k,2:4))
            aa=[aa;matriz(k,1)];
            break
        end
        k=k+1;
    end
end
aa(1:end,2)=' ';
aa=[aa,aa];
aamut=aa(1:end,2:4);

function [ind,aamut,codonesmut,codonesori,condmut] =
defmutfold(codori,codmut,mutotal)
matriz =
['FTTT';'FTTC';'LTTA';'LTTG';'LCTT';'LCTC';'LCTA';'LCTG';'IATT';'IATC';'IATA';
'MATG';'VGTT';'VGTC';'VGTA';'VGTG';'STCT';'STCC';'STCA';...
'STCG';'SAGT';'SAGC';'PCCT';'PCCC';'PCCA';'PCCG';'TACT';'TACC';'TACA';'TACG';'
AGCT';'AGCC';'AGCA';'AGCG';'YTAT';'YTAC';'HCAT';'HCAC';...
'QCAA';'QCAG';'NAAT';'NAAC';'KAAA';'KAAG';'DGAT';'DGAC';'EGAA';'EGAG';'CTGT';'
CTGC';'WTGG';'RCGT';'RCGC';'RCGA';'RCGG';'RAGA';'RAGG';...
        'GGGT';'GGGC';'GGGA';'GGGG';'XTAA';'XTAG';'XTGA'];
ind=[];
codonesmut=[];
codonesori=[];
aamut=[];
aa=[];
condmut=[];
i=1;
while i <= length(mutotal(:,1))
    ind=[ind,mutotal(i,1)];
    codonesmut=[codonesmut;codmut(mutotal(i,1),1:3)];
    codonesori=[codonesori;codori(mutotal(i,1),1:3)];
    condmut=[condmut,mutotal(i,2)];
    i=i+1;
end
for j = 1:length(ind)
    k=1;
    while k <= 64
        if strcmpi(codonesmut(j,1:3),matriz(k,2:4))
            aa=[aa;matriz(k,1)];
            break
        end
        k=k+1;
    end
end
aa(1:end,2)=' ';
aa=[aa,aa];
aamut=aa(1:end,2:4);
end

function [numcodones] = detfoldcod(foldind,maxfold)
numcodones=[];
for i = 1:length(foldind(:,1))
    if length(foldind(i,1):foldind(i,2))>maxfold
        cod1 = 0;
        cod2 = 0;
        cod3 = 0;
        cod4 = 0;
        ubi1 = mod(foldind(i,1),3);
```

```
        ubi2 = mod(foldind(i,2),3);
        ubi3 = mod(foldind(i,3),3);
        ubi4 = mod(foldind(i,4),3);
        if ubi1 == 0
            cod1=foldind(i,1)/3+1;
        elseif ubi1 == 1
            cod1=(foldind(i,1)+2)/3;
        elseif ubi1 == 2
            if (foldind(i,2)-foldind(i,1))>= 4
                cod1=(foldind(i,1)+1)/3+1;
            else
                cod1=(foldind(i,1)+1)/3;
            end
        end
        if ubi2 == 0
            cod2=foldind(i,2)/3;
        elseif ubi2 == 1
            cod2=(foldind(i,2)-1)/3;
        elseif ubi2 == 2
            cod2=(foldind(i,2)-2)/3;
        end
        if ubi3 == 0
            cod3=foldind(i,3)/3+1;
        elseif ubi3 == 1
            cod3=(foldind(i,3)+2)/3;
        elseif ubi3 == 2
            if (foldind(i,4)-foldind(i,3))>= 4
                cod3=(foldind(i,3)+1)/3+1;
            else
                cod3=(foldind(i,3)+1)/3;
            end
        end
        if ubi4 == 0
            cod4=foldind(i,4)/3;
        elseif ubi4 == 1
            cod4=(foldind(i,4)-1)/3;
        elseif ubi4 == 2
            cod4=(foldind(i,4)-2)/3;
        end
        numcodones=[numcodones;cod1,cod2,cod3,cod4];
    else
        break
    end
end

function [filgene] = filcode(factor,codgene)
lcode=length(codgene);
if factor == 1
    indini=strfind(codgene,'atg');
    if ~isempty(indini)
        testmax=length(codgene(indini(end):end));
        seqs1=[];
        seqs2=[];
        for i = 1:length(indini);
            if length(codgene(indini(i):end))>=testmax
                j=indini(i)+3;
                indend=[];
                while j <= lcode-2
```

```
                    if
strcmpi([codgene(j),codgene(j+1),codgene(j+2)],'taa')||strcmpi([codgene(j),cod
gene(j+1),codgene(j+2)],'tag')||...
strcmpi([codgene(j),codgene(j+1),codgene(j+2)],'tga')
                        indend=j+2;
                        break
                    end
                    j=j+3;
                end
                if ~isempty(indend)
                    seqs1=[seqs1,length(codgene(indini(i):indend))];
                    testmax = max(seqs1);
                    if testmax == length(codgene(indini(i):indend));
                        filgene=codgene(indini(i):indend);
                    end
                else
                    seqs2=[seqs2,length(codgene(indini(i):end))];
                    if max(seqs2)== length(codgene(indini(i):end));
                        posfilgene=codgene(indini(i):end);
                    end
                end
            else
                break
            end
        end
        if isempty(seqs1)
            comp = mod(length(posfilgene),3);
            if comp == 0
                filgene=posfilgene;
            else
                filgene=posfilgene(1:end-comp);
            end
        end
    else
        comp = mod(lcode,3);
        if comp == 0
            filgene=codgene;
        else
            filgene=codgene(1:lcode-comp);
        end
    end
elseif factor == 0
    comp = mod(lcode,3);
    if comp == 0
        filgene=codgene;
    else
        filgene=codgene(1:lcode-comp);
    end
end

function [filind] = filtroind(newind,oldind)
ncod=length(newind);
filind=[];
for i = 1:ncod
    if isempty(find(oldind==newind(i)))
        filind=[filind,newind(i)];
    else
        continue
    end
```

```
end

function [energia] = foldenergy(sec)
eini=[0.98,1.03];
pairnn=['aa';'tt';'at';'ta';'ca';'tg';'gt';'ac';'ct';'ag';'ga';'tc';'cg';'gc';
'gg';'cc'];
valpairnn=[-1,-1,-0.88,-0.58,-1.45,-1.45,-1.44,-1.44,-1.28,-1.28,-1.3,-1.3,-
2.17,-2.24,-1.84,-1.84];
a1=length(pairnn);
a2=length(valpairnn);
energia=0;
largo=length(sec);
if strcmpi(sec(1),'c')||strcmpi(sec(1),'g')
    energia=energia+eini(1);
else
    energia=energia+eini(2);
end
if strcmpi(sec(largo),'c')||strcmpi(sec(largo),'g')
    energia=energia+eini(1);
else
    energia=energia+eini(2);
end
for i=1:largo-1
    j=1;
    while j <= length(pairnn)
        if strcmpi(sec(i:i+1),pairnn(j,1:2))
            energia=energia+valpairnn(j);
            break
        end
        j=j+1;
    end
end

function [mfold] = foldtest(foldcodigo)
codigo=lower(foldcodigo);
largo=length(codigo);
ind1=[];
ind2=[];
indini=[];
i=1;
while i <= largo-11
    ls=3;
    sec=codigo(i:i+ls);
    secobj=seqcomplement(fliplr(sec));
    indini=strfind(codigo((i+8):end),secobj);
    if ~isempty(indini)
        indini=indini+i+7;
        ind1=[ind1,i,indini];
        for j=1:length(indini)
            ind2=[ind2;i,indini(j)];
        end
    end
i=i+1;
end
lind2=length(ind2(:,1));
ind3=[];
ind4=[];
k=1;
while k <= lind2
```

```
    postar=[];
    if ~isempty(find(ind2(:,1)==ind2(k,1)+1))
        cont=4;
        pos1=find(ind2(:,1)==ind2(k,1)+1);
            if ~isempty(find(ind2(pos1(1):pos1(1)+length(pos1)-
1,2)==ind2(k,2)-1))
                postar=pos1(find(ind2(pos1(1):pos1(1)+length(pos1)-
1,2)==ind2(k,2)-1));
                cont=cont+1;
                ind2(postar,:)=[];
                lind2=lind2-1;
                n=k;
                m=2;
                while m<=max(ind2(:,1))
                    if ~isempty(find(ind2(n:end,1)==ind2(k,1)+m))
                        pos2=find(ind2(:,1)==ind2(k,1)+m);
                        if ~isempty(find(ind2(pos2(1):pos2(1)+length(pos2)-
1,2)==ind2(k,2)-m))

postar=pos2(find(ind2(pos2(1):pos2(1)+length(pos2)-1,2)==ind2(k,2)-m));
                            ind2(postar,:)=[];
                            lind2=lind2-1;
                            cont=cont+1;
                        else
                            ind4=[ind4;ind2(k,1),ind2(k,2)-m+1,cont];
                            k=k+1;
                            break
                        end
                    else
                        ind4=[ind4;ind2(k,1),ind2(k,2)-m+1,cont];
                        k=k+1;
                        break
                    end
                   m=m+1;
                end
            else
                ind3=[ind3;ind2(k,:)];
                k=k+1;
            end
    else
       ind3=[ind3;ind2(k,:)];
       k=k+1;
    end
end

gener1=[];
gener2=[];
for p = 1:length(ind4(:,1))
    gener1=[gener1,foldenergy(codigo(ind4(p,1):ind4(p,1)+ind4(p,3)-1))];
end
ind3(:,3)=4;
for p = 1:length(ind3(:,1))
    gener2=[gener2,foldenergy(codigo(ind3(p,1):ind3(p,1)+ind3(p,3)-1))];
end
ind4(:,4)=gener1';
ind3(:,4)=gener2';
indene=sortrows([ind4;ind3],4);
mfold=[indene(1,1),indene(1,1)+indene(1,3)-
1,indene(1,2),indene(1,2)+indene(1,3)-1,indene(1,4)];
```

```
if length(indene(:,1))>1
    for i=2:length(indene(:,1))
        j=1;
        while j<=length(mfold(:,1))
            if ((indene(i,1)<mfold(j,1)&&(indene(i,2)+indene(i,3)-
1)<mfold(j,1))||...
                ((indene(i,1)+indene(i,3)-
1)<mfold(j,1)&&indene(i,2)>mfold(j,4))||...
                (indene(i,1)>mfold(j,2)&&(indene(i,2)+indene(i,3)-
1)<mfold(j,3))||...
                (indene(i,1)>mfold(j,4)))
                if j==length(mfold(:,1));
                    mfold=[mfold;indene(i,1),indene(i,1)+indene(i,3)-
1,indene(i,2),indene(i,2)+indene(i,3)-1,indene(i,4)];
                    break
                end
                j=j+1;
            else
                break
            end
        end
    end
end

function [codigomc] = mjcodon(codigomod,codigocs,mcodon)
codigomc=[];
for i=1:length(codigomod)
    if strcmpi(codigomod(i,1:3),codigocs(i,1:3))
        codigomc=[codigomc;codigomod(i,1:3)];
    else
        j=1;
        while j<=length(mcodon)
            if strcmpi(codigocs(i,1:3),mcodon(j,1:3))
                codigomc=[codigomc;codigocs(i,1:3)];
                break
            end
            if j == length(mcodon)
                codigomc=[codigomc;codigomod(i,1:3)];
            end
            j=j+1;
        end
    end
end
function [mjcodon] = mjvector(tinfo,ocodon)
mjcodon=[];
i=1;
while i <= 63
    if ~strcmpi(ocodon(i,1),'m')&&~strcmpi(ocodon(i,1),'w')
        maxval=tinfo(i);
        ind=i;
        for j = i:63
            if strcmpi(ocodon(j,1),ocodon(j+1,1))
                if maxval>= tinfo(j+1)
                    continue
                else
                    ind=j+1;
                    maxval=tinfo(j+1);
                end
            else
```

```
                mjcodon=[mjcodon;ocodon(ind,2:4)];
                i=j+1;
                break
            end
        end
    else
        i=i+1;
    end
    if j == 63
        mjcodon=[mjcodon;ocodon(ind,2:4)];
        break
    end
end

function [codigo] = mutationer(codigogen,codonest,codonescu,ramp)
totalcod = floor(length(codigogen));
mutciclo=floor(totalcod*0.02);
codaa = codigoaa(codigogen);
codigo = codigogen;
ciclo1=1;
ciclo2=1;
i=1;
j=1;
while ciclo2 <= totalcod || ciclo1 <= totalcod
    ncodon=[];
    matriz=[];
    codones=[];
    mutkin=[];
    codigostr=[];
    [ncodon,matriz] = cusagemut(codigo);
    pcodon = ncodon/totalcod;
    for z=1:length(codigo)
        codigostr=[codigostr,codigo(z,1:3)];
    end
    codones = valcodones(codigostr,matriz,pcodon);
    [mutkin] = mutdet(codones,codonescu,codonest,ramp);
    mutcodon = zeros(1,64);
    for m = 1:64
        n=1;
        while n<=length(codigo)
            if strcmpi(matriz(m,2:4),codigo(n,1:3))
                mutcodon(m)=mutkin(n);
                break
            end
            n=n+1;
        end
    end
    indmut2=find(mutkin(1:end)==2);
    indmut1=find(mutkin(1:end)==1);
    cbreak=0;
    if ciclo2>=indmut2(end) && ciclo1>=indmut1(end)
        break
    elseif ciclo2>=indmut2(end) && isempty(indmut1)
        break
    elseif ciclo1>=indmut1(end) && isempty(indmut2)
        break
    elseif isempty(indmut2) && isempty(indmut1)
        break
    else
```

```
        if ~isempty(indmut1) && (ciclo2>=indmut2(end)||isempty(indmut2))
            if i ~= 1
                ind1=find(indmut1>ciclo1);
                i=ind1(1);
            end
            while i <= length(indmut1) && cbreak<=mutciclo
                codpomut=[];
                valpomut=[];
                porpomut=[];
                cc=[];
                minmut=[];
                aa=codaa(indmut1(i));
                indsin=find(matriz(1:end,1)==aa);
                k=1;
                while k <= length(indsin)
                    if strcmpi(aa,matriz(indsin(k),1)) &&
~strcmpi(codigo(indmut1(i),1:3),matriz(indsin(k),2:4))
                        if mutcodon(indsin(k))==3 || mutcodon(indsin(k))==4
                            codpomut = [codpomut;matriz(indsin(k),2:4)];
                            valpomut = [valpomut,mutcodon(indsin(k))];
                            porpomut = [porpomut,pcodon(indsin(k))];
                        end
                    end
                    k=k+1;
                end
                if length(valpomut)==1
                    codigo(indmut1(i),1:3) = codpomut;
                    cbreak=cbreak+1;
                elseif length(valpomut)>1
                    if ~isempty(find(valpomut==4))
                        cc = find(valpomut==4);
                        for p = 1:length(cc)
                            minmut=[minmut,porpomut(cc(p))];
                        end
                        posol=find(porpomut==min(minmut));
                        if length(posol)>1
                            codopt=[];
                            for r = 1:length(posol)
                                if
strcmpi(codpomut(posol(r),1:3),codigo(find(codaa(1:(indmut1(i)-
1))==aa,1,'last'),1:3))
                                    codopt=codpomut(posol(r),1:3);
                                    break
                                end
                            end
                            if ~isempty(codopt)
                                codigo(indmut1(i),1:3) = codopt;
                                cbreak=cbreak+1;
                            else
                                codigo(indmut1(i),1:3) =
codpomut(find(porpomut==min(minmut),1,'first'),1:3);
                                cbreak=cbreak+1;
                            end
                        else
                            codigo(indmut1(i),1:3) = codpomut(posol,1:3);
                            cbreak=cbreak+1;
                        end
                    else
                        posol=find(porpomut==min(porpomut));
```

```
                        if length(posol)>1
                            codopt=[];
                            for r = 1:length(posol)
                                if
strcmpi(codpomut(posol(r),1:3),codigo(find(codaa(1:(indmut1(i)-
1))==aa,1,'last'),1:3))
                                    codopt=codpomut(posol(r),1:3);
                                    break
                                end
                            end
                            if ~isempty(codopt)
                                codigo(indmut1(i),1:3) = codopt;
                                cbreak=cbreak+1;
                            else
                                codigo(indmut1(i),1:3) =
codpomut(find(porpomut==min(porpomut),1,'first'),1:3);
                                cbreak=cbreak+1;
                            end
                        else
                            codigo(indmut1(i),1:3) = codpomut(posol,1:3);
                            cbreak=cbreak+1;
                        end
                    end
                end
                i=i+1;
            end
            if i >= 2
            ciclo1=indmut1(i-1);
            end
        elseif ~isempty(indmut2) && ciclo2~=indmut2(end)
            if j ~= 1
                ind2=find(indmut2>ciclo2);
                j=ind2(1);
            end
            while j <= length(indmut2) && cbreak<=mutciclo
                codpomut=[];
                valpomut=[];
                porpomut=[];
                cc=[];
                minmut=[];
                aa=codaa(indmut2(j));
                indsin=find(matriz(1:end,1)==aa);
                k=1;
                while k <= length(indsin)
                    if strcmpi(aa,matriz(indsin(k),1)) &&
~strcmpi(codigo(indmut2(j),1:3),matriz(indsin(k),2:4))
                        if mutcodon(indsin(k))==3 || mutcodon(indsin(k))==4
                            codpomut = [codpomut;matriz(indsin(k),2:4)];
                            valpomut = [valpomut,mutcodon(indsin(k))];
                            porpomut = [porpomut,pcodon(indsin(k))];
                        end
                    end
                    k=k+1;
                end
                if length(valpomut)==1
                    codigo(indmut2(j),1:3) = codpomut;
                    cbreak=cbreak+1;
                elseif length(valpomut)>1
                    if ~isempty(find(valpomut==4))
```

```
                        cc = find(valpomut==4);
                        for p = 1:length(cc)
                            minmut=[minmut,porpomut(cc(p))];
                        end
                        posol=find(porpomut==min(minmut));
                        if length(posol)>1
                            codopt=[];
                            for r = 1:length(posol)
                                if
strcmpi(codpomut(posol(r),1:3),codigo(find(codaa(1:(indmut2(j)-
1))==aa,1,'last'),1:3))
                                    codopt=codpomut(posol(r),1:3);
                                    break
                                end
                            end
                            if ~isempty(codopt)
                                codigo(indmut2(j),1:3) = codopt;
                                cbreak=cbreak+1;
                            else
                                codigo(indmut2(j),1:3) =
codpomut(find(porpomut==min(minmut),1,'first'),1:3);
                                cbreak=cbreak+1;
                            end
                        else
                            codigo(indmut2(j),1:3) = codpomut(posol,1:3);
                            cbreak=cbreak+1;
                        end
                    else
                        posol=find(porpomut==min(porpomut));
                        if length(posol)>1
                            codopt=[];
                            for r = 1:length(posol)
                                if
strcmpi(codpomut(posol(r),1:3),codigo(find(codaa(1:(indmut2(j)-
1))==aa,1,'last'),1:3))
                                    codopt=codpomut(posol(r),1:3);
                                    break
                                end
                            end
                            if ~isempty(codopt)
                                codigo(indmut2(j),1:3) = codopt;
                                cbreak=cbreak+1;
                            else
                                codigo(indmut2(j),1:3) =
codpomut(find(porpomut==min(porpomut),1,'first'),1:3);
                                cbreak=cbreak+1;
                            end
                        else
                            codigo(indmut2(j),1:3) = codpomut(posol,1:3);
                            cbreak=cbreak+1;
                        end
                    end
                end
                j=j+1;
            end
            if j >= 2
            ciclo2=indmut2(j-1);
            end
        end
```

```
    end
end

function [mut] = mutdet(codones,codonescu,codonest,ramp)
largo=length(codones);
i=1;
mut=[];
while i <= largo
    if codones(i) >= 0 && codones(i) < 0.01
        if codonest(i) >= 0 && codonest(i) < 0.01
            if i<=ramp
                mut=[mut,5];
                i=i+1;
                continue
            else
                if codonescu(i) >= 0 && codonescu(i) < 0.01
                    mut=[mut,1];
                    i=i+1;
                    continue
                elseif codonescu(i) >= 0.01 && codonescu(i) < 0.02
                    mut=[mut,1];
                    i=i+1;
                    continue
                elseif codonescu(i) >= 0.02
                    mut=[mut,1];
                    i=i+1;
                    continue
                end
            end
        elseif codonest(i) >= 0.01 && codonest(i) < 0.02
            if codonescu(i) >= 0 && codonescu(i) < 0.01
                if
(codones(i)+0.0025)<=codonest(i)&&(codonescu(i)+0.0075)>codones(i)
                    mut=[mut,3];
                else
                    mut=[mut,5];
                end
                i=i+1;
                continue
            elseif codonescu(i) >= 0.01 && codonescu(i) < 0.02
                mut=[mut,3];
                i=i+1;
                continue
            elseif codonescu(i) >= 0.02
                mut=[mut,3];
                i=i+1;
                continue
            end
        elseif codonest(i) >= 0.02
            if codonescu(i) >= 0 && codonescu(i) < 0.01
                mut=[mut,3];
                i=i+1;
                continue
            elseif codonescu(i) >= 0.01 && codonescu(i) < 0.02
                mut=[mut,4];
                i=i+1;
                continue
            elseif codonescu(i) >= 0.02
                mut=[mut,4];
```

```
                i=i+1;
                continue
            end
        end
    elseif codones(i) >= 0.01 && codones(i) < 0.02
        if codonest(i) >= 0 && codonest(i) < 0.01
            if i<=ramp
                mut=[mut,5];
                i=i+1;
                continue
            else
                if codonescu(i) >= 0 && codonescu(i) < 0.01
                    mut=[mut,1];
                    i=i+1;
                    continue
                elseif codonescu(i) >= 0.01 && codonescu(i) < 0.02
                    mut=[mut,1];
                    i=i+1;
                    continue
                elseif codonescu(i) >= 0.02
                    mut=[mut,1];
                    i=i+1;
                    continue
                end
            end
        elseif codonest(i) >= 0.01 && codonest(i) < 0.02
            if codonescu(i) >= 0 && codonescu(i) < 0.01
                if
(codones(i)+0.0025)<=codonest(i)&&(codonescu(i)+0.0075)>codones(i)
                    mut=[mut,3];
                else
                    mut=[mut,5];
                end
                i=i+1;
                continue
            elseif codonescu(i) >= 0.01 && codonescu(i) < 0.02
                if codones(i)<=(codonest(i)+0.005)
                    mut=[mut,3];
                else
                    mut=[mut,5];
                end
                i=i+1;
                continue
            elseif codonescu(i) >= 0.02
                if codones(i)<=(codonest(i)+0.005)
                    mut=[mut,3];
                else
                    mut=[mut,5];
                end
                i=i+1;
                continue
            end
        elseif codonest(i) >= 0.02
            if codonescu(i) >= 0 && codonescu(i) < 0.01
                if
(codones(i)+0.0025)<=codonest(i)&&(codonescu(i)+0.0075)>codones(i)
                    mut=[mut,3];
                else
                    mut=[mut,5];
```

```
            end
            i=i+1;
            continue
        elseif codonescu(i) >= 0.01 && codonescu(i) < 0.02
            mut=[mut,3];
            i=i+1;
            continue
        elseif codonescu(i) >= 0.02
            mut=[mut,3];
            i=i+1;
            continue
        end
    end
elseif codones(i) >= 0.02
    if codonest(i) >= 0 && codonest(i) < 0.01
        if i<=ramp
            mut=[mut,5];
            i=i+1;
            continue
        else
            if codonescu(i) >= 0 && codonescu(i) < 0.01
                mut=[mut,2];
                i=i+1;
                continue
            elseif codonescu(i) >= 0.01 && codonescu(i) < 0.02
                mut=[mut,2];
                i=i+1;
                continue
            elseif codonescu(i) >= 0.02
                mut=[mut,2];
                i=i+1;
                continue
            end
        end
    elseif codonest(i) >= 0.01 && codonest(i) < 0.02
        if codonescu(i) >= 0 && codonescu(i) < 0.01
            mut=[mut,2];
            i=i+1;
            continue
        elseif codonescu(i) >= 0.01 && codonescu(i) < 0.02
            mut=[mut,1];
            i=i+1;
            continue
        elseif codonescu(i) >= 0.02
            if codones(i)<=(codonest(i)+0.005)
                mut=[mut,3];
            else
                mut=[mut,5];
            end
            i=i+1;
            continue
        end
    elseif codonest(i) >= 0.02
        if codonescu(i) >= 0 && codonescu(i) < 0.01
            if codones(i)<=codonest(i)
                mut=[mut,1];
            else
                mut=[mut,2];
            end
```

```
                i=i+1;
                continue
            elseif codonescu(i) >= 0.01 && codonescu(i) < 0.02
                if codones(i)<=(codonest(i)+0.005)
                    mut=[mut,3];
                else
                    mut=[mut,5];
                end
                i=i+1;
                continue
            elseif codonescu(i) >= 0.02
                if codones(i)<=(codonest(i)+0.005)
                    mut=[mut,3];
                else
                    mut=[mut,5];
                end
                i=i+1;
                continue
            end
        end
    end
end

function [codigo,mutotal] =
mutfoldcod(codigostr,codonest,codonescu,foldind,maxfold,codxfil,ramp)
codonfold = detfoldcod(foldind,maxfold);
codigo=lower(codigostr);
totalcod = length(codigo);
codaa = codigoaa(codigo);
mutotal=[];
for i = 1:length(codonfold(:,1))
    if isempty(codxfil)
indcodon=[codonfold(i,1):codonfold(i,2),codonfold(i,3):codonfold(i,4)];
    else
indcodon=filtroind([codonfold(i,1):codonfold(i,2),codonfold(i,3):codonfold(i,4
)],codxfil);
    end
    ncodon=[];
    matriz=[];
    codones=[];
    mutkin=[];
    codestr=[];
    [ncodon,matriz] = cusagemut(codigo);
    pcodon = ncodon/totalcod;
    for j=1:length(codigo)
        codestr=[codestr,codigo(j,1:3)];
    end
    codones = valcodones(codestr,matriz,pcodon);
    [mutkin] = mutdet(codones,codonescu,codonest,ramp);
    mutcodon = zeros(1,64);
    for m = 1:64
        n=1;
        while n<=length(codigo)
            if strcmpi(matriz(m,2:4),codigo(n,1:3))
                mutcodon(m)=mutkin(n);
                break
            end
            n=n+1;
        end
```

```
    end
    indmut=[];
    for k = 1:length(indcodon)
       indmut=[indmut,mutkin(indcodon(k))];
    end
    codmut=[];
    if ~isempty(find(indmut==1))
        codmut=[codmut,indcodon(find(indmut==1))];
    end
    if ~isempty(find(indmut==2))
        codmut=[codmut,indcodon(find(indmut==2))];
    end
    if ~isempty(find(indmut==5))
        codmut=[codmut,indcodon(find(indmut==5))];
    end
    if ~isempty(find(indmut==3))
        codmut=[codmut,indcodon(find(indmut==3))];
    end
    if ~isempty(find(indmut==4))
        codmut=[codmut,indcodon(find(indmut==4))];
    end
    mut=[];
    cond=1;
    while cond
        for j = 1:length(codmut)
            codpomut=[];
            valpomut=[];
            porpomut=[];
            cc=[];
            minmut=[];
            aa=codaa(codmut(j));
            indsin=find(matriz(1:end,1)==aa);
            if ~isempty(indsin)
                k=1;
                while k <= length(indsin)
                    if strcmpi(aa,matriz(indsin(k),1)) &&
~strcmpi(codigo(codmut(j),1:3),matriz(indsin(k),2:4))
                        if mutcodon(indsin(k))==3 || mutcodon(indsin(k))==4
                            codpomut = [codpomut;matriz(indsin(k),2:4)];
                            valpomut = [valpomut,mutcodon(indsin(k))];
                            porpomut = [porpomut,pcodon(indsin(k))];
                        end
                    end
                    k=k+1;
                end
                if length(valpomut)==1
                    mut=[codmut(j),1];
                    codestr(codmut(j)*3-2:codmut(j)*3) = codpomut;
                    codigo(codmut(j),1:3) = codpomut;
                    break
                elseif length(valpomut)>1
                    if ~isempty(find(valpomut==4))
                        minmut=porpomut(find(valpomut==4));
                        posol=find(porpomut==min(minmut));
                        if length(posol)>1
                            codopt=[];
                            for r = 1:length(posol)
```

```
                                if
strcmpi(codpomut(posol(r),1:3),codigo(find(codaa(1:(codmut(j)-
1))==aa,1,'last'),1:3))
                                    codopt=codpomut(posol(r),1:3);
                                    break
                                end
                            end
                            if ~isempty(codopt)
                                mut=[codmut(j),1];
                                codestr(codmut(j)*3-2:codmut(j)*3) = codopt;
                                codigo(codmut(j),1:3) = codopt;
                                break
                            else
                                mut=[codmut(j),1];
                                codestr(codmut(j)*3-2:codmut(j)*3) =
codpomut(find(porpomut==min(minmut),1,'first'),1:3);
                                codigo(codmut(j),1:3) =
codpomut(find(porpomut==min(minmut),1,'first'),1:3);
                                break
                            end
                        else
                            mut=[codmut(j),1];
                            codestr(codmut(j)*3-2:codmut(j)*3) =
codpomut(posol,1:3);
                            codigo(codmut(j),1:3) = codpomut(posol,1:3);
                            break
                        end
                    else
                        posol=find(porpomut==min(porpomut));
                        if length(posol)>1
                            codopt=[];
                            for r = 1:length(posol)
                                if
strcmpi(codpomut(posol(r),1:3),codigo(find(codaa(1:(codmut(j)-
1))==aa,1,'last'),1:3))
                                    codopt=codpomut(posol(r),1:3);
                                    break
                                end
                            end
                            if ~isempty(codopt)
                                mut=[codmut(j),1];
                                codestr(codmut(j)*3-2:codmut(j)*3) = codopt;
                                codigo(codmut(j),1:3) = codopt;
                                break
                            else
                                mut=[codmut(j),1];
                                codestr(codmut(j)*3-2:codmut(j)*3) =
codpomut(find(porpomut==min(porpomut),1,'first'),1:3);
                                codigo(codmut(j),1:3) =
codpomut(find(porpomut==min(porpomut),1,'first'),1:3);
                                break
                            end
                        else
                            mut=[codmut(j),1];
                            codestr(codmut(j)*3-2:codmut(j)*3) =
codpomut(posol,1:3);
                            codigo(codmut(j),1:3) = codpomut(posol,1:3);
                            break
                        end
```

```
                    end
                end
            end
        end
        if isempty(mut)
            for j = 1:length(codmut)
                codpomut=[];
                valpomut=[];
                porpomut=[];
                cc=[];
                minmut=[];
                aa=codaa(codmut(j));
                indsin=find(matriz(1:end,1)==aa);
                if ~isempty(indsin)
                    k=1;
                    while k <= length(indsin)
                        if strcmpi(aa,matriz(indsin(k),1)) &&
~strcmpi(codigo(codmut(j),1:3),matriz(indsin(k),2:4))
                            if mutcodon(indsin(k))==5
                                codpomut = [codpomut;matriz(indsin(k),2:4)];
                                valpomut = [valpomut,mutcodon(indsin(k))];
                                porpomut = [porpomut,pcodon(indsin(k))];
                            end
                        end
                        k=k+1;
                    end
                    if length(valpomut)==1
                        mut=[codmut(j),2];
                        codestr(codmut(j)*3-2:codmut(j)*3) = codpomut;
                        codigo(codmut(j),1:3) = codpomut;
                        break
                    elseif length(valpomut)>1
                        posol=find(porpomut==min(porpomut));
                        if length(posol)>1
                            codopt=[];
                            for r = 1:length(posol)
                                if
strcmpi(codpomut(posol(r),1:3),codigo(find(codaa(1:(codmut(j)-
1))==aa,1,'last'),1:3))
                                    codopt=codpomut(posol(r),1:3);
                                    break
                                end
                            end
                            if ~isempty(codopt)
                                mut=[codmut(j),2];
                                codestr(codmut(j)*3-2:codmut(j)*3) = codopt;
                                codigo(codmut(j),1:3) = codopt;
                                break
                            else
                                mut=[codmut(j),2];
                                codestr(codmut(j)*3-2:codmut(j)*3) =
codpomut(find(porpomut==min(porpomut),1,'first'),1:3);
                                codigo(codmut(j),1:3) =
codpomut(find(porpomut==min(porpomut),1,'first'),1:3);
                                break
                            end
                        else
                            mut=[codmut(j),2];
```

```
                            codestr(codmut(j)*3-2:codmut(j)*3) =
codpomut(posol,1:3);
                            codigo(codmut(j),1:3) = codpomut(posol,1:3);
                            break
                        end
                    end
                end
            end
            if isempty(mut)
                for j = 1:length(codmut)
                    codpomut=[];
                    valpomut=[];
                    porpomut=[];
                    cc=[];
                    minmut=[];
                    aa=codaa(codmut(j));
                    indsin=find(matriz(1:end,1)==aa);
                    if ~isempty(indsin)
                        k=1;
                        while k <= length(indsin)
                            if strcmpi(aa,matriz(indsin(k),1)) &&
~strcmpi(codigo(codmut(j),1:3),matriz(indsin(k),2:4))
                                if mutcodon(indsin(k))==1 ||
mutcodon(indsin(k))==2
                                    codpomut =
[codpomut;matriz(indsin(k),2:4)];
                                    valpomut = [valpomut,mutcodon(indsin(k))];
                                    porpomut = [porpomut,pcodon(indsin(k))];
                                end
                            end
                            k=k+1;
                        end
                        if length(valpomut)==1
                            mut=[codmut(j),3];
                            codestr(codmut(j)*3-2:codmut(j)*3) = codpomut;
                            codigo(codmut(j),1:3) = codpomut;
                            break
                        elseif length(valpomut)>1
                            if ~isempty(find(valpomut==1))
                                minmut=porpomut(find(valpomut==1));
                                posol=find(porpomut==min(minmut));
                                if length(posol)>1
                                    codopt=[];
                                    for r = 1:length(posol)
                                        if
strcmpi(codpomut(posol(r),1:3),codigo(find(codaa(1:(codmut(j)-
1))==aa,1,'last'),1:3))
codopt=codpomut(posol(r),1:3);
                                            break
                                        end
                                    end
                                    if ~isempty(codopt)
                                        mut=[codmut(j),3];
                                        codestr(codmut(j)*3-2:codmut(j)*3) =
codopt;
                                        codigo(codmut(j),1:3) = codopt;
                                        break
                                    else
                                        mut=[codmut(j),3];
```

```
                                        codestr(codmut(j)*3-2:codmut(j)*3) =
codpomut(find(porpomut==min(minmut),1,'first'),1:3);
                                        codigo(codmut(j),1:3) =
codpomut(find(porpomut==min(minmut),1,'first'),1:3);
                                        break
                                    end
                                else
                                    mut=[codmut(j),3];
                                    codestr(codmut(j)*3-2:codmut(j)*3) =
codpomut(posol,1:3);
                                    codigo(codmut(j),1:3) =
codpomut(posol,1:3);
                                    break
                                end
                            else
                                posol=find(porpomut==min(porpomut));
                                if length(posol)>1
                                    codopt=[];
                                    for r = 1:length(posol)
                                        if
strcmpi(codpomut(posol(r),1:3),codigo(find(codaa(1:(codmut(j)-
1))==aa,1,'last'),1:3))
codopt=codpomut(posol(r),1:3);
                                            break
                                        end
                                    end
                                    if ~isempty(codopt)
                                        mut=[codmut(j),3];
                                        codestr(codmut(j)*3-2:codmut(j)*3) =
codopt;
                                        codigo(codmut(j),1:3) = codopt;
                                        break
                                    else
                                        mut=[codmut(j),3];
                                        codestr(codmut(j)*3-2:codmut(j)*3) =
codpomut(find(porpomut==min(porpomut),1,'first'),1:3);
                                        codigo(codmut(j),1:3) =
codpomut(find(porpomut==min(porpomut),1,'first'),1:3);
                                        break
                                    end
                                else
                                    mut=[codmut(j),3];
                                    codestr(codmut(j)*3-2:codmut(j)*3) =
codpomut(posol,1:3);
                                    codigo(codmut(j),1:3) =
codpomut(posol,1:3);
                                    break
                                end
                            end
                        end
                    end
                end
            end
        end
        mutotal=[mutotal;mut];
        [cond,codmut]= testnumfold(foldind(i,:),codestr,maxfold,codxfil);
        mut=[];
    end
end
```

```
function [linea] = pregraf(indices,largo)
linea=[];
for i=1:largo
    linea=strcat(linea,'.');
end
for j = 1:length(indices(:,1))
    linea(indices(j,1):indices(j,2))='(';
    linea(indices(j,3):indices(j,4))=')';
end
end

function codgene = rcode(codigo)
fid = fopen(codigo,'r');
if fid == -1
    codgene = [];
else
    precode = [];
    fila = 1;
    while ~feof(fid)
        linea = fgetl(fid);
        i=1;
        if linea(i) ~= '>'
            precode=strcat(precode,linea);
        elseif linea(i) == '>'
            codnom(fila,:) = linea(i+1);
            fila = fila + 1;
        end
    end
    fclose(fid);
    codgene=lower(precode);
end

function [ffold,codonesxmut] = testnumfold(vector,codigo,maxfold,codxfil)
sec=codigo(vector(1):vector(2));
secobj=seqcomplement(fliplr(sec));
secinv=codigo(vector(3):vector(4));
n=0;
count=0;
countmax=0;
nvector=[];
for i = 1:length(sec)
    if strcmpi(secobj(i),secinv(i))
        n=n+1;
        count=n;
        if count >= 2 && count > countmax
           countmax=count;
            nvector=[vector(2)-i+1,vector(2)+countmax-i,...
                vector(3)+i-countmax,vector(3)+i-1];
        end
    else
        n=0;
    end
end
codonesxmut = [];
if ~isempty(nvector) && length(codigo(nvector(1):nvector(2)))> maxfold
    numcodones = detfoldcod(nvector,maxfold);
    if isempty(codxfil)
```

```
        codonesxmut =
[numcodones(1):numcodones(2),numcodones(3):numcodones(4)];
        ffold=1;
    else
        codonesxmut =
filtroind([numcodones(1):numcodones(2),numcodones(3):numcodones(4)],codxfil);
        if ~isempty(codonesxmut)
            ffold=1;
        else
            ffold=0;
        end
    end
else
    ffold=0;
end

function [ind,sequence] = trna_check(cond,cugen)
matriz=[0   0.02204 0.01102 0.01102 0   0.01102 0.00924 0.0398  0   0.02625 0
0.10601 0   0.02204 0.05511 0   0   0.02204 0.01102 0.01102 0   0.01102 0
0.01102 0.01242 0.01102 0   0.02042 0.01403 0.01429 0   0.0206  0.02937 0   0
0.03307 0   0.01102 0.01847 0.02204 0   0.04409 0.06479 0   0   0.03307
0.05091 0   0   0.01102 0.01102 0.04409 0   0.01015 0.01102 0.04221 0.01102 0
0.04409 0.01403 0.01102 0   0   0.01102;...
        0   0.03488 0.03488 0.01163 0   0.01163 0.02326 0.01163 0   0.03488 0
0.06977 0   0.01163 0.04651 0   0   0.01163 0.02326 0   0   0.02326 0   0
0.03488 0   0   0.01163 0.04651 0   0   0.01163 0.05814 0   0   0.02326 0
0.02326 0.04651 0   0   0.04651 0.04651 0   0   0.04651 0.06977 0   0
0.01163 0.01163 0.04651 0   0   0.01163 0.01163 0.01163 0   0.04651 0.03488 0
0   0   0;...
        0   0.03833 0.02439 0.03484 0   0.00348 0.01045 0   0.0453  0.00348
0.00697 0.03833 0.04878 0   0.01045 0.00697 0.03833 0   0.01394 0.00348 0
0.00697 0.00697 0   0.03484 0   0.03833 0   0.01742 0.00348 0.03833 0
0.02091 0   0   0.02787 0   0.02787 0.03136 0.00348 0   0.03833 0.02787
0.04878 0   0.05575 0.05226 0.00697 0   0.01394 0.02091 0.02439 0   0
0.00348 0.04181 0.00348 0   0.05575 0.01045 0.00697 0   0   0.00348;...
        0   0.01606 0.00803 0.00803 0.01004 0   0.00803 0.02008 0.0261  0
0.01004 0.03213 0.0261  0   0.01004 0.01807 0.01205 0   0.00602 0.00602 0
0.01807 0.01406 0   0.01205 0.00602 0.01606 0.00201 0.00803 0.00803 0.04418
0.00201 0.03012 0.01205 0   0.02209 0   0.02811 0.01004 0.02209 0   0.03414
0.03012 0.17269 0   0.03012 0.01406 0.04016 0   0.08233 0.01205 0.01205 0
0.01205 0.00602 0.01004 0.01205 0.00201 0.02811 0.01205 0.01406 0   0
0.00402;...
        0   0.02425 0.0091  0.01031 0.0144  0   0.00697 0.02546 0.02954 0
0.00576 0.04181 0.04747 0   0.00697 0.04673 0.02062 0   0.00576 0.01635 0
0.01728 0.02016 0   0.01802 0.02406 0.02471 0   0.01245 0.01106 0.0406  0
0.00697 0.01226 0   0.04162 0   0.03372 0.01152 0.02666 0   0.03335 0.0197
0.04227 0   0.04757 0.02332 0.04989 0   0.01774 0.02062 0.0288  0   0.02638 0
0.00743 0.00743 0   0.04153 0.0197  0   0   0   0.00167;...
        0   0.02328 0.01358 0.01358 0.02328 0   0.00582 0.0194  0.02716
0.01064 0.0097  0.0388  0.02134 0   0.0097  0.03104 0.02134 0   0.0097
0.00776 0   0.01552 0.0194  0   0.01358 0.00776 0.0194  0   0.01164 0.01164
0.05626 0   0.01746 0.0097  0.00194 0.02716 0   0.02134 0.02134 0.03976
0.00292 0.06208 0.032   0.03298 0   0.03686 0.02522 0.02522 0   0.0582
0.01746 0.01358 0   0.01164 0.00872 0.01164 0.0097  0   0.0291  0.01746
0.01358 0.00388 0.00194 0.00582;...
        0.00238 0.03333 0.00476 0.01429 0.01429 0.00238 0.0119  0.0119
0.02857 0   0.00952 0.05714 0.03095 0   0.00476 0.02619 0.03095 0   0.0119
0.01667 0   0.02143 0.03095 0   0.01905 0.00952 0.02381 0   0.0119  0.01429
0.04524 0   0.02143 0.01905 0   0.01905 0   0.02143 0.02143 0.02381 0.00238
```

```
0.03333 0.01905 0.0381  0   0.02619 0.01667 0.03333 0   0.01667 0.02143
0.01905 0.00238 0.01667 0.0119  0.01429 0.02381 0   0.03571 0.02143 0.02619 0
0.00476 0.00238;...
        0.00028 0.02765 0.0058  0.01821 0.02396 0   0.01279 0.01417 0.04278
0.00104 0.00637 0.07094 0.02323 0.00567 0.00604 0.016   0.01573 0.00986
0.01494 0.00775 0   0.02053 0.02011 0   0.02402 0.01074 0.01709 0.00609 0.0142
0.00827 0.03899 0   0.01811 0.01826 0.0011  0.0227  0.00242 0.02648 0.02006
0.01688 0.00058 0.03064 0.03409 0.05521 0.00143 0.03839 0.02122 0.03145
0.00028 0.02168 0.02419 0.02655 0.00052 0.00688 0.00885 0.01112 0.01355 0
0.03705 0.01355 0.01125 0   0.0017  0.00058;...
        0   0.03436 0.02532 0.01989 0.01808 0   0.01627 0.01266 0.0217  0
0.01627 0.04702 0.02712 0   0.00904 0.01989 0.0217  0.00181 0.02351 0.00542 0
0.01447 0.0217  0   0.02712 0.00723 0.01808 0.00362 0.01627 0.00542 0.01989 0
0.08499 0.00904 0.00181 0.03255 0.00181 0.02351 0.01447 0.01989 0.00181
0.03797 0.01989 0.02712 0.00362 0.04159 0.02351 0.03436 0   0.01989 0.01808
0.01989 0   0.00904 0.00723 0.01085 0.01447 0   0.03074 0.0217  0.01085
0.00362 0   0.00181];
ind=0;
if cond == 0
    j=1;
    totalvar=[];
    pcugen=cugen/sum(cugen);
    while j <= 9
        sumvar=0;
        for i=1:64
            sumvar=sumvar+((pcugen(i)-
(pcugen(i)+matriz(j,i))/2)^2+(matriz(j,i)-(pcugen(i)+matriz(j,i))/2)^2);
        end
        totalvar(j)=sumvar;
        minvar = min(totalvar);
        if minvar == totalvar(j)
            ind=j;
        end
        j=j+1;
    end
    sequence=matriz(ind,:);
else
    ind = cond;
    sequence=matriz(ind,:);
end
end

function [codones] = valcodones(filcodegen,ocodon,pcodon)
lcode=length(filcodegen);
j=1;
codones=[];
while j<=(lcode-2)
    k=1;
    while k<=64
    if strcmpi(filcodegen(j:(j+2)),ocodon(k,2:4))
    codones=[codones,pcodon(k)];
    break
    end
    k=k+1;
    end
    j=j+3;
end
end
```

5.4 Ventanas Programa

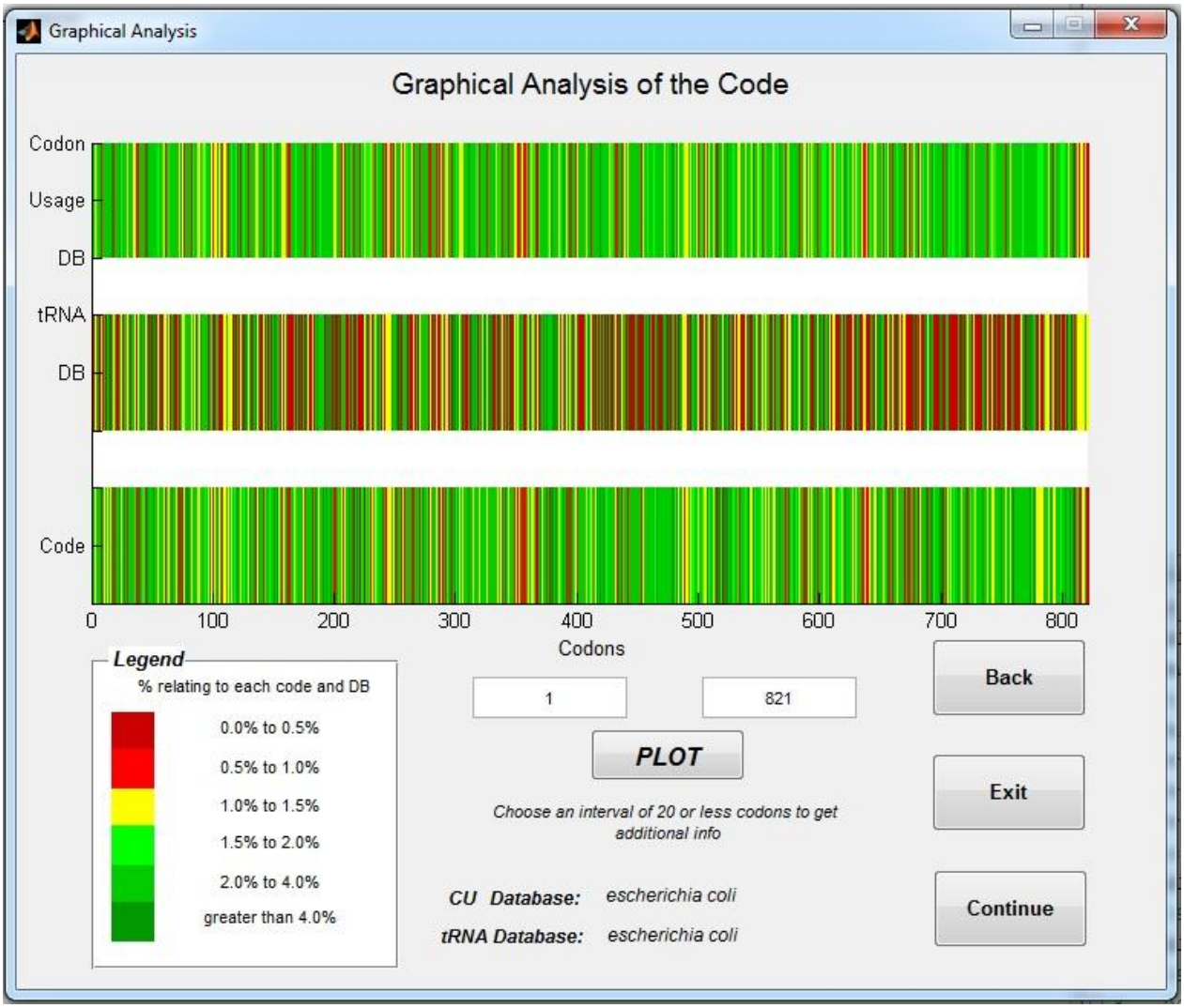

Figura 10: Ventana 3: Análisis grafico general de la secuencia

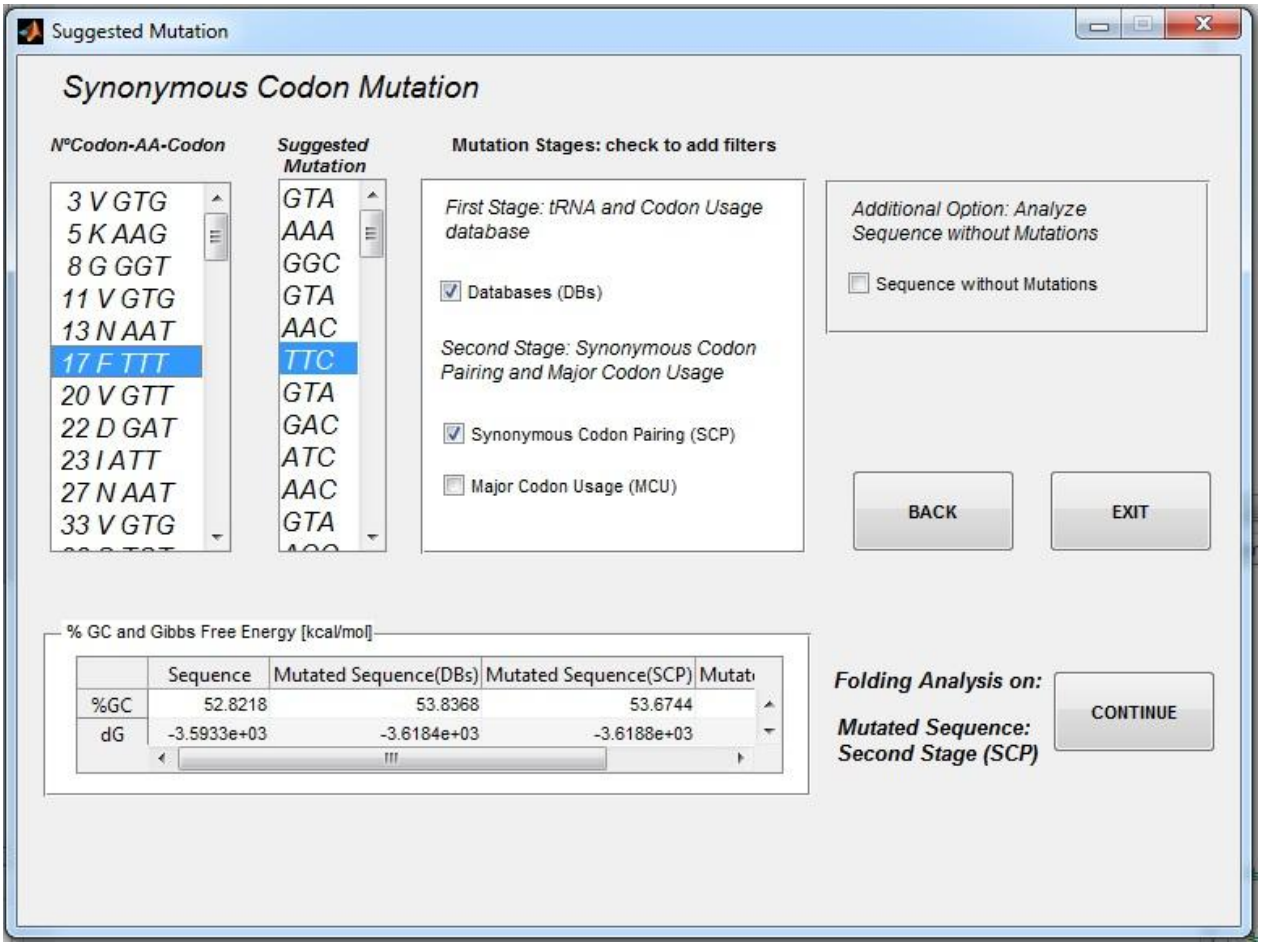

Figura 11: Ventana 4: Mutaciones sugeridas sin Rampa

5.5 Secuencias

5.5.1 eGFP

eGFP WT

ATGCGTAAAGGAGAAGAACTTTTCACTGGAGTTGTCCCAATTCTTGTTGAATTAGAT
GGTGATGTTAATGGGCACAAATTTTCTGTCAGTGGAGAGGGTGAAGGTGATGCAAC
ATACGGAAAACTTACCCTTAAATTTATTTGCACTACTGGAAAACTACCTGTTCCATG
GCCAACACTTGTCACTACTTTCGGTTATGGTGTTCAATGCTTTGCGAGATACCCAGA
TCATATGAAACAGCATGACTTTTTCAAGAGTGCCATGCCCGAAGGTTATGTACAGG
AAAGAACTATATTTTTCAAAGATGACGGGAACTACAAGACACGTGCTGAAGTCAAGT
TTGAAGGTGATACCCTTGTTAATAGAATCGAGTTAAAAGGTATTGATTTTAAAGAAG
ATGGAAACATTCTTGGACACAAATTGGAATACAACTATAACTCACACAATGTATACA
TCATGGCAGACAAACAAAAGAATGGAATCAAAGTTAACTTCAAAATTAGACACAACA
TTGAAGATGGAAGCGTTCAACTAGCAGACCATTATCAACAAAATACTCCAATTGGCG
ATGGCCCTGTCCTTTTACCAGACAACCATTACCTGTCCACACAATCTGCCCTTTCGA
AAGATCCCAACGAAAAGAGAGACCACATGGTCCTTCTTGAGTTTGTAACAGCTGCT
GGGATTACACATGGCATGGATGAACTATACAAAAGGCCTGCAGCAAACGACGAAAA
CTACGCTGCAGCAGTTTAA

eGFP high expression table (HT)
ATGAGAAAGGGTGAAGAATTGTTCACAGGTGTGGTGCCAATTTTGGTGGAATTGGA
TGGTGATGTGAATGGTCATAAGTTCTCAGTGTCAGGTGAAGGTGAAGGTGATGCTA
CATACGGTAAGTTGACATTGAAGTTCATTTGTACAACAGGTAAGTTGCCAGTGCCAT
GGCCAACATTGGTGACAACATTCGGTTACGGTGTGCAATGTTTCGCTAGATACCCA
GATCATATGAAGCAACATGATTTCTTCAAGTCAGCTATGCCAGAAGGTTACGTGCAA
GAAAGAACAATTTTCTTCAAGGATGATGGTAATTACAAGACAAGAGCTGAAGTGAAG
TTCGAAGGTGATACATTGGTGAATAGAATTGAATTGAAGGGTATTGATTTCAAGGAA
GATGGTAATATTTTGGGTCATAAGTTGGAATACAATTACAATTCACATAATGTGTACA
TTATGGCTGATAAGCAAAAGAATGGTATTAAGGTGAATTTCAAGATTAGACATAATA
TTGAAGATGGTTCAGTGCAATTGGCTGATCATTACCAACAAAATACACCAATTGGTG
ATGGTCCAGTGTTGTTGCCAGATAATCATTACTTGTCAACACAATCAGCTTTGTCAA
AGGATCCAAATGAAAAGAGAGATCATATGGTGTTGTTGGAATTCGTGACAGCTGCT
GGTATTACACATGGTATGGATGAATTGTACAAGAGACCAGCTGCTAATGATGAAAAT
TACGCTGCTGCTGTGTAA

eGFP control matrix 2 (C2)
ATGAGAAAAGGTGAAGAATTGTTCACAGGAGTTGTTCCCATTTTAGTTGAATTAGAC
GGTGATGTAAATGGTCATAAATTTTCTGTTTCCGGCGAAGGAGAGGGAGATGCAAC
GTACGGAAAGTTGACATTGAAGTTTATATGCACCACAGGAAAGCTTCCCGTTCCAT
GGCCTACCTTGGTAACGACGTTTGGTTATGGTGTTCAATGCTTTGCTCGATATCCG
GATCACATGAAGCAGCATGATTTCTTCAAGAGCGCTATGCCCGAAGGGTATGTTCA
AGAAAGAACCATTTTCTTTAAAGATGATGGCAATTATAAGACAAGAGCTGAAGTAAA
ATTCGAGGGAGATACATTGGTTAATCGAATAGAATTAAAGGGTATTGACTTTAAAGA
GGATGGTAATATTCTGGGTCACAAACTTGAATACAATTATAATTCCCATAACGTCTA
CATAATGGCCGACAAACAAAAGAATGGTATTAAAGTCAATTTCAAAATTCGTCATAA
CATCGAGGACGGCAGCGTCCAATTAGCGGATCACTATCAACAAAATACACCTATAG
GTGACGGTCCCGTGCTTTTACCAGACAACCACTATCTAAGCACTCAATCTGCTCTAT
CGAAAGATCCTAACGAAAAAAGAGATCATATGGTGCTGTTAGAATTTGTCACTGCG
GCTGGTATTACACATGGGATGGACGAACTTTACAAAAGGCCCGCTGCTAATGATGA
AAATTATGCAGCCGCCGTTTAA

eGFP con las mutaciones sugeridas por el programa
ATGCGTAAAGGAGAAGAACTTTTCACTGGAGTTGTCCCAATTCTTGTTGAATTAGAT
GGTGATGTTAATGGGCACAAATTTTCTGTCAGTGGAGAAGGTGAAGGTGATGCAAC
TTACGGAAAATTAACATTAAAATTCATTTGCACTACTGGAAAACTACCAGTTCCATG
GCCAACACTAGTAACTACTTTCGGTTACGGTGTTCAATGCTTCGCTAGATACCCAGA
TCATATGAAACAACATGACTTCTTCAAGTCAGCCATGCCAGAAGGTTATGTACAAGA
AAGAACTATTTTCTTCAAAGATGACGGGAACTACAAGACTCGTGCTGAAGTTAAGTT
CGAAGGTGATACTTTAGTTAATAGAATTGAGTTAAAAGGTATTGATTTCAAAGAAGA
TGGAAACATTTTAGGACACAAATTGGAATACAACTATAACTCACACAATGTATACATT
ATGGCAGACAAACAAAAGAATGGAATTAAAGTTAACTTCAAAATTAGACACAACATT
GAAGATGGATCTGTTCAACTAGCAGACCATTATCAACAAAATACTCCAATTGGCGAT
GGCCCAGTTTTATTACCAGACAACCATTACCTGTCTACACAATCTGCCTTATCTAAA
GATCCAAACGAAAAGAGAGACCACATGGTTTTATTAGAGTTTGTAACAGCTGCTGG
GATTACACATGGCATGGATGAACTATACAAACGTCCAGCAGCAAACGACGAAAACT
ACGCTGCAGCAGTTTAA

5.5.2 ADN polimerasa

ADN Polimerasa WT

ATGAAGCATATGCCGAGAAAGATGTATAGTTGTGACTTTGAGACAACTACTAAAGTG
GAAGACTGTAGGGTATGGGCGTATGGTTATATGAATATAGAAGATCACAGTGAGTA
CAAAATAGGTAATAGCCTGGATGAGTTTATGGCGTGGGTGTTGAAGGTACAAGCTG
ATCTATATTTCCATAACCTCAAATTTGACGGAGCTTTTATCATTAACTGGTTGGAACG
TAATGGTTTTAAGTGGTCGGCTGACGGATTGCCAAACACATATAATACGATCATATC
TCGCATGGGACAATGGTACATGATTGATATATGTTTAGGCTACAAAGGGAAACGTA
AGATACATACAGTGATATATGACAGCTTAAAGAAACTACCGTTTCCTGTTAAGAAGA
TAGCTAAAGACTTTAAACTAACTGTTCTTAAAGGTGATATTGATTACCACAAAGAAAG
ACCAGTCGGCTATAAGATAACACCCGAAGAATACGCCTATATTAAAAACGATATTCA
GATTATTGCGGAACGTCTGTTAATTCAGTTTAAGCAAGGTTTAGACCGGATGACAG
CAGGCAGTGACAGTCTAAAAGGTTTCAAGGATATTATAACCACTAAGAAATTCAAAA
AGGTGTTTCCTACATTGAGTCTTGGACTCGATAAGGAAGTGAGATACGCCTATAGA
GGTGGTTTTACATGGTTAAATGATAGGTTCAAAGAAAAAGAAATCGGAGAAGGCAT
GGTCTTCGATGTTAATAGTCTATATCCTGCACAGATGTATAGCCGTCTCCTTCCATA
TGGTGAACCTATAGTATTCGAGGGTAAATACGTTTGGGACGAAGATTACCCACTAC
ACATACAGCATATCAGATGTGAGTTCGAATTGAAAGAGGGCTATATACCCACTATAC
AGATAAAAAGAAGTAGGTTTTATAAAGGTAATGAGTACCTAAAAAGTAGCGGCGGG
GAGATAGCCGACCTCTGGTTGTCAAATGTAGACCTAGAATTAATGAAAGAACACTA
CGATTTATATAACGTTGAATATATCAGCGGCTTAAAATTTAAAGTAACTACAGGTTTG
TTTAAAGATTTTATAGATAAATGGACGTACATCAAGACGACATCAGAAGGAGCGATC
AAGCAACTAGCAAAACTGATGTTAAACAGTCTATACGGTAAATTCGCTAGTAACCCT
GATGTTACAGGGAAAGTCCCTTATTTAAAAGAGAATGGGGCGCTAGGTTTCAGACT
TGGAGAAGAGGAAACAAAAGACCCTGTTTATACACCTATGGGCGTTTTCATCACTG
CATGGGCTAGATACACGACAATTACAGCGGCACAGGCTTGTTATGATCGGATAATA
TACTGTGATACTGACAGCATACATTTAACGGGTACAGAGATACCTGATGTAATAAAA
GATATAGTTGACCCTAAGAAATTGGGATACTGGGCACATGAAAGTACATTCAAAAGA
GCTAAATATCTGAGACAGAAGACCTATATACAAGACATCTATATGAAAGAAGTAGAT
GGTAAGTTAGTAGAAGGTAGTCCAGATGATTACACTGATATAAAATTTAGTGTTAAA
TGTGCGGGAATGACTGACAAGATTAAGAAAGAGGTTACGTTTGAGAATTTCAAAGT
CGGATTCAGTCGGAAAATGAAGCCTAAGCCTGTGCAAGTGCCGGGCGGGGTGGTT
CTGGTTGATGACACATTCACAATCAAATAA

ADN Polimerasa P19

ATGAAACATATGCCACGCAAAATGTACTCGTGCGACTTCGAAACCACCACCAAGGT
TGAGGATTGCCGTGTTTGGGCTTACGGCTACATGAACATCGAGGACCATTCCGAAT
ATAAGATCGGCAACTCTTTAGACGAATTTATGGCTTGGGTTCTCAAAGTTCAGGCG
GACTTGTACTTTCACAATTTGAAGTTCGATGGTGCGTTCATTATCAATTGGCTCGAG
CGCAACGGCTTCAAATGGAGCGCGGATGGCCTCCCGAATACCTACAACACCATTAT
CAGCCGTATGGGTCAGTGGTATATGATCGACATCTGCCTGGGTTATAAGGGTAAGC
GCAAAATCCACACCGTTATCTACGATTCTCTGAAAAAGTTGCCATTCCCGGTGAAAA
AAATCGCGAAGGATTTCAAGTTGACCGTGTTGAAGGGCGACATCGACTATCATAAG
GAGCGTCCGGTTGGTTACAAAATCACCCCGGAGGAGTATGCGTACATCAAGAATGA
CATCCAAATCATCGCTGAGGCGTTACTGATCCAATTCAAACAGGGCCTGGATCGTA
TGACCGCGGGTTCCGATTCCTTGAAGGGCTTTAAAGACATCATCACTACCAAAAAG
TTTAAGAAAGTTTTCCCGACCCTCTCCTTGGGTTTGGACAAAGAGGTTCGTTATGCG

TACCGTGGCGGCTTCACCTGGCTGAACGACCGTTTTAAGGAGAAGGAGATTGGTG
AGGGTATGGTTTTTGACGTGAACTCCTTGTACCCGGCGCAAATGTACTCCCGCTTG
TTGCCGTACGGCGAGCCGATCGTTTTTGAAGGCAAGTATGTGTGGGATGAGGACT
ATCCGTTGCATATCCAACACATTCGTTGCGAATTTGAGCTCAAGGAAGGTTACATCC
CGACCATCCAAATCAAGCGTTCCCGTTTCTACAAGGGCAACGAATATTTGAAGTCC
TCTGGTGGTGAAATCGCGGATTTGTGGCTCAGCAACGTTGATTTGGAGCTGATGAA
GGAGCATTATGACCTGTACAATGTGGAGTACATTTCTGGTCTGAAGTTCAAGGCGA
CCACCGGCCTCTTCAAGGACTTCATCGACAAGTGGACCTATATTAAAACCACCAGC
GAGGGTGCTATTAAACAGTTGGCGAAGTTAATGCTGAACTCCTTGTATGGCAAGTT
TGCGTCCAATCCGGACGTGACCGGTAAGGTTCCGTACCTGAAGGAAAACGGTGCT
TTGGGCTTTCGTTTGGGTGAGGAAGAGACTAAGGATCCGGTGTACACCCCGATGG
GTGTGTTTATTACCGCGTGGGCGCGTTATACCACCATCACCGCTGCGCAAGCGTG
CTACGACCGTATCATCTATTGCGACACCGATTCTATCCACCTGACCGGCACCGAAA
TCCCGGACGTTATCAAGGACATCGTGGATCCGAAAAAGCTCGGTTATTGGGCGCA
CGAGTCCACCTTTAAGCGTGCGAAGTACTTACGTCAAAAAACTTACATCCAGGATAT
TTACATGAAGGAGGTTGACGGCAAACTGGTTGAGGGCTCCCCGGACGACTATACC
GACATCAAGTTCTCCGTGAAGTGCGCTGGTATGACCGATAAAATCAAAAAGGAAGT
GACCTTCGAAAACTTTAAGGTTGGTTTTTCCCGTAAGATGAAACCGAAACCGGTTCA
GGTTCCAGGTGGTGTTGTGTTAGTGGACGATACCTTTACCATTAAG

ADN Polimerasa P15

ATGAAACATATGCCGCGTAAAATGTATTCTTGTGACTTTGAAACCACTACTAAAGTG
GAAGACTGTCGTGTTTGGGCGTATGGTTATATGAACATCGAAGATCACTCTGAATA
CAAAATCGGTAACTCCCTGGATGAATTTATGGCGTGGGTGCTGAAAGTTCAGGCTG
ATCTGTATTTCCATAACCTGAAATTTGACGGTGCTTTTATCATTAACTGGCTGGAAC
GTAACGGTTTTAAATGGTCTGCTGACGGTCTGCCGAACACCTATAACACCATCATCT
CTCGCATGGGTCAGTGGTACATGATTGATATCTGTCTGGGCTACAAAGGTAAACGT
AAAATCCATACCGTGATCTATGACTCCCTGAAAAAACTGCCGTTTCCGGTTAAAAAA
ATCGCTAAAGACTTTAAACTGACTGTTCTGAAAGGTGATATTGATTACCACAAAGAA
CGTCCGGTTGGCTATAAAATCACCCCGGAAGAATACGCGTATATTAAAAACGATATT
CAGATTATTGCGGAAGCTCTGCTGATTCAGTTTAAACAGGGTCTGGACCGTATGAC
CGCGGGCTCTGACTCTCTGAAAGGTTTCAAAGATATTATCACCACTAAAAAATTCAA
AAAAGTGTTTCCGACCCTGTCTCTGGGTCTGGATAAAGAAGTGCGTTACGCGTATC
GTGGTGGTTTTACCTGGCTGAACGATCGTTTCAAAGAAAAAGAAATCGGTGAAGGC
ATGGTTTTCGATGTTAACTCTCTGTATCCGGCGCAGATGTATTCTCGTCTGCTGCCG
TATGGTGAACCGATCGTTTTCGAAGGTAAATACGTTTGGGACGAAGATTACCCGCT
GCACATCCAGCATATCCGTTGTGAATTTGAACTGAAAGAAGGCTATATCCCGACTAT
CCAGATCAAACGTTCTCGTTTTTATAAAGGTAACGAATACCTGAAATCTTCCGGCGG
TGAAATCGCGGACCTGTGGCTGTCTAACGTTGACCTGGAACTGATGAAAGAACACT
ACGATCTGTATAACGTTGAATATATCTCCGGCCTGAAATTTAAAGCGACTACCGGTC
TGTTTAAAGATTTTATTGATAAATGGACCTACATCAAAACCACCTCTGAAGGTGCGA
TCAAACAGCTGGCGAAACTGATGCTGAACTCTCTGTACGGTAAATTCGCTTCTAAC
CCGGATGTTACCGGTAAAGTTCCGTATCTGAAAGAAAACGGTGCGCTGGGTTTCCG
TCTGGGTGAAGAAGAAACCAAAGACCCGGTTTATACCCCGATGGGCGTTTTCATCA
CTGCGTGGGCTCGTTACACCACCATTACCGCGGCGCAGGCTTGTTATGATCGTATC
ATCTACTGTGATACTGACTCCATCCATCTGACCGGTACCGAAATCCCGGATGTTATC
AAAGATATCGTTGACCCGAAAAAACTGGGTTACTGGGCGCATGAATCTACCTTCAA
ACGTGCTAAATATCTGCGTCAGAAAACCTATATCCAGGACATCTATATGAAAGAAGT

TGATGGTAAACTGGTTGAAGGTTCTCCGGATGATTACACTGATATCAAATTTTCTGT
TAAATGTGCGGGTATGACTGACAAAATTAAAAAAGAAGTTACCTTTGAAAACTTCAA
AGTTGGTTTCTCTCGTAAAATGAAACCGAAACCGGTGCAGGTGCCGGGCGGTGTG
GTTCTGGTTGATGACACCTTCACCATCAAA

ADN Polimerasa con las mutaciones sugeridas por el programa
ATGAAGCACATGCCGAGAAAGATGTACAGCTGTGACTTCGAAACCACGACGAAAGT
AGAAGACTGTAGGGTATGGGCCTACGGGTACATGAACATCGAAGACCACAGCGAA
TACAAAATAGGCAACAGCCTGGACGAGTTTATGGCCTGGGTATTGAAGGTACAAGC
CGACCTGTATTTCCATAACCTCAAATTTGACGGAGCATTTATCATTAACTGGTTGGA
ACGTAATGGCTTTAAGTGGTCGGCAGACGGATTGCCAAACACCTATAATACGATCA
TATCGCGTATGGGACAATGGTACATGATTGACATATGTCTGGGCTACAAAGGGAAA
CGTAAGATACATACCGTGATATATGACAGCCTGAAGAAACTGCCGTTTCCGGTCAA
GAAGATAGCAAAAGACTTTAAACTGACTGTCCTGAAAGGCGACATTGATTACCACAA
AGAAAGACCAGTCGGCTATAAGATAACCCCCGAAGAATACGCCTATATTAAAAACG
ATATTCAGATTATTGCGGAACGTCTGTTAATTCAGTTTAAGCAAGGTTTAGACCGGA
TGACCGCAGGCTCGGACTCGCTCAAAGGTTTCAAGGATATTATAACCACTAAGAAA
TTCAAAAAGGTGTTTCCCACCTTGTCGCTCGGACTCGATAAGGAAGTGAGATACGC
CTATAGAGGTGGTTTTACCTGGTTAAATGATAGGTTCAAAGAAAAAGAAATCGGAGA
AGGCATGGTCTTCGATGTTAATTCGCTCTATCCCGCACAGATGTATAGCCGTCTCC
TCCCATATGGTGAACCCATAGTATTCGAGGGTAAATACGTTTGGGACGAAGATTAC
CCACTCCACATACAGCATATCAGATGTGAGTTCGAATTGAAAGAGGGCTATATACC
CACTATACAGATAAAAAGATCAAGGTTTTATAAAGGTAATGAGTACTTGAAATCAAG
CGGCGGGGAGATAGCCGACCTCTGGTTGTCAAATGTAGACTTGGAATTAATGAAAG
AACACTACGATTTATATAACGTTGAATATATCAGCGGCTTAAAATTTAAAGTAACTAC
CGGTTTGTTTAAAGATTTTATAGATAAATGGACGTACATCAAGACGACCTCAGAAGG
AGCGATCAAGCAATTGGCAAAACTGATGTTAAACAGTTTGTACGGTAAATTCGCTAG
TAACCCGGATGTTACAGGGAAAGTCCCGTATTTAAAAGAGAATGGGGCGTTGGGTT
TCAGATTGGGAGAAGAGGAAACAAAAGACCCGGTTTATACACCGATGGGCGTTTTC
ATCACTGCATGGGCTAGATACACGACAATTACAGCGGCACAGGCTTGTTATGATCG
GATAATATACTGTGATACTGACAGCATACATTTAACGGGTACAGAGATACCGGATGT
AATAAAAGATATAGTTGACCCGAAGAAACTCGGATACTGGGCACATGAAAGTACATT
CAAAAGAGCTAAATATCTGAGACAGAAGACCTATATACAAGACATCTATATGAAAGA
AGTAGATGGTAAGTTAGTAGAAGGTAGTCCAGATGATTACACTGATATAAAATTTAG
TGTTAAATGTGCGGGAATGACTGACAAGATTAAGAAAGAGGTTACGTTTGAGAATTT
CAAAGTCGGATTCAGTCGGAAAATGAAGCCAAAGCCAGTGCAAGTGCCGGGCGGG
GTGGTTCTGGTTGATGACACATTCACAATCAAATAA