UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
<75.12> ANÁLISIS NUMÉRICO

DATOS DEL TRABAJO PRÁCTICO

1	2016	Análisis de errores
	AÑO	y estabilidad en algoritmos
	1	para la resolución del mismo problema
TP NRO	CUAT	TEMA

INTEGRANTES DEL GRUPO

7	Merlo Leiva Nahuel	92115
	APELLIDO Y NOMBRE	PADRÓN
	Cozza Fabrizio Luis	97402
GRUPO	APELLIDO Y NOMBRE	PADRÓN

DATOS DE LA ENTREGA

C314.TXT	58-1698	04052016	04052016
ARCHIVO	NRO CONTROL	FECHA VENC	FECHA ENTR

CORRECCIONES

FECHA	NOTA	OBSERVACIONES
DOCENTE	FIRMA	

Introducción

En la actualidad, científicos e ingenieros trabajan en problemas cada vez más complejos. En general se requiere el uso de computadoras para estudiar modelos matemáticos a través del uso de la algoritmia.

Mediante esta metodología, una computadora es capaz de calcular aproximaciones a la solución de un problema determinado. Debido a que la aritmética con que opera tiene precisión finita, pueden introducirse errores de redondeo y truncamiento en las operaciones efectuadas.

Entonces, es inevitable pensar en la utilización de algoritmos simples para reducir el tiempo de cómputo, uso de memoria como también de errores, con la posible contradicción de no obtener un resultado, quizás, más preciso.

Objetivos

- Implementar algoritmos propuestos que permitan calcular la resolución del problema del movimiento de un cuerpo celeste respecto de otro, mediante los hallazgos de Newton.
- Interpretar el cálculo, desarrollo y resultado del problema presentado teniendo en cuenta en el análisis teórico sus errores relativos, estabilidad y orden de los algoritmos presentados.

Resumen

En primer lugar, se desarrollarán los algoritmos propuestos para aplicar la resolución al problema original propuesto por Newton.

Se desarrollarán en total dos algoritmos para calcular el movimiento del pequeño planeta volcánico Mustafar, situado en un sistema homónimo, y próximo a su estrella.

Durante el desarrollo se realizaran cálculos del llamado desvío real relativo y el tiempo de corrida del algoritmo entre otros parámetros no tan principales como estos, aumentando cada vez el “paso”, siendo cada uno al menos un orden de magnitud superior al previo.

Finalmente, se procederá a buscar una solución aproximada para cada algoritmo, considerando las particularidades de propagación de errores que podrían presentar su desarrollo, y obteniendo conclusiones respecto a su precisión, orden y estabilidad, a través del análisis teórico e interpretación de gráficos.

A - B - C: Sistema Mustafar y algoritmos propuestos

La ley de gravitación de Newton propone una solución al problema del movimiento de cuerpos celestes, siendo este acotado a un plano.

Utilizando un sistema de coordenadas polares con origen en uno de los cuerpos, el movimiento del otro cuerpo puede ser representado por las siguientes ecuaciones:

$$\frac{d^2u}{d\theta} + u = \frac{\mu}{h^2} \quad (1)$$

$$u = \frac{1}{r} \ , \mu = G\ (m_1 + m_2) = GM \quad (2)$$

$$\theta = 0, \frac{du}{d\theta} = 0 \quad (3)$$

La idea es mostrar el modelo original planteado por Newton para luego mostrar las soluciones por medio de algoritmia, independientemente de los parámetros especificados por Newton.

Los algoritmos a desarrollar se calcularan con simple precisión y utilizan los siguientes parámetros:

$$\lambda = \text{Nº Padrón del alumno} \ / \ \ 9.10^4$$

$$m_1 = \ \lambda\, 1.9891\ 10^{30}\ kg$$

$$m_2 = \ \lambda\, 3.301\ 10^{23}\ kg$$

$$e = \lambda^{-1}\, 0.2056$$

$$a = \ \lambda^2\, 5.791\ 10^7\ km$$

$$M = m_1 + \ m_2$$

$$h^2 = aGM(1 - e^2)\ m^4 \ / \ s^2$$

Y u_n la aproximación numérica de $u(\theta_n)$, θ_n=n k, k=2π/N y N el número de pasos.

Se utilizó el lenguaje C++ para la programación de los algoritmos con las siguientes características del PC:

Procesador: Intel(R) Core(TM) i5-2500K

Velocidad de reloj: 3,30 GHz (3,70 GHz turbo)

Cantidad de núcleos: 4

Sistema operativo: Windows 10 Pro 64 Bits

En cuanto a su desarrollo está dado por los siguientes pseudocódigos:

Algoritmo 1

$u_0 = a^{-1}\,(1-e)^{-1}$

$v_0 = 0$

Desde $n = 0$ a N - 1

$u_{n+1} = u_n + k v_n$

$v_{n+1} = v_n - k u_n + k\mu h^{-2}$

Avanzar n

Algoritmo 2

$u_0 = a^{-1}\,(1-e)^{-1}$

$v_0 = 0$

$\alpha = \mu h^{-2}$

Desde $n = 0$ a N - 1

$w_1 = u_n + k\,v_n\,/\,2$

$z_1 = v_n + k\,(\alpha - u_n)\,/\,2$

$w_2 = u_n + k\,z_1\,/\,2$

$z_2 = v_n + k\,(\alpha - w_1)\,/\,2$

$w_3 = u_n + k\,z_2$

$z_3 = v_n + k\,(\alpha - w_2)\,/\,2$

$u_{n+1} = u_n + k\,(v_n + 2\,z_1 + 2\,z_2 + z_3)\,/\,6$

$v_{n+1} = v_n + k\,(6\alpha - u_n - 2\,w_1 - 2\,w_2 - w_3)\,/\,6$

Avanzar n

B. 3 – B.4: Desvío real relativo y tabla de resultados

Por medio de la siguiente formula se pasa a calcular el desvío real relativo (DRR):

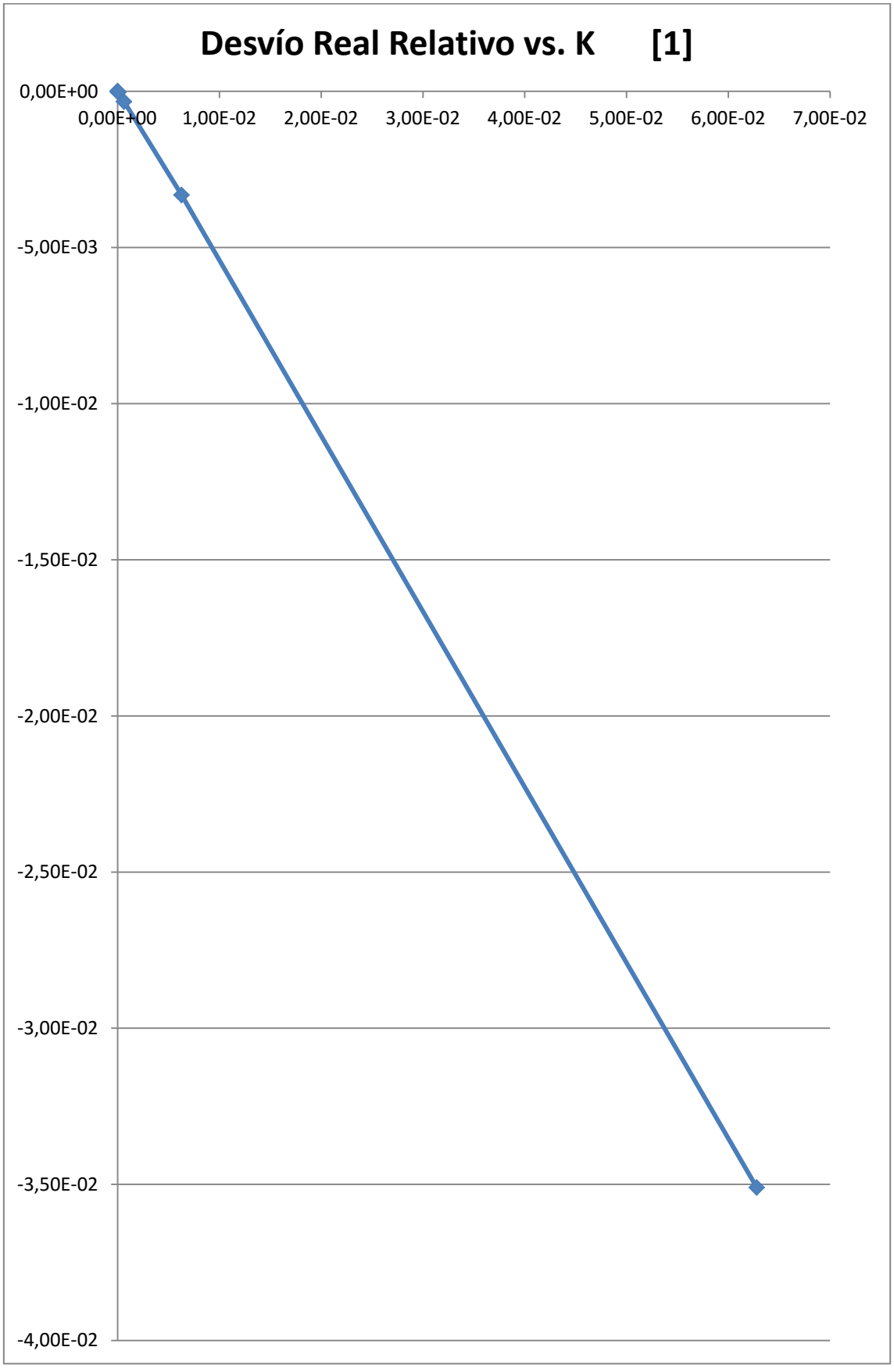
$$\Delta r(N)/\, q = [\, {u_n}^{-1} \,/\, q] - 1$$

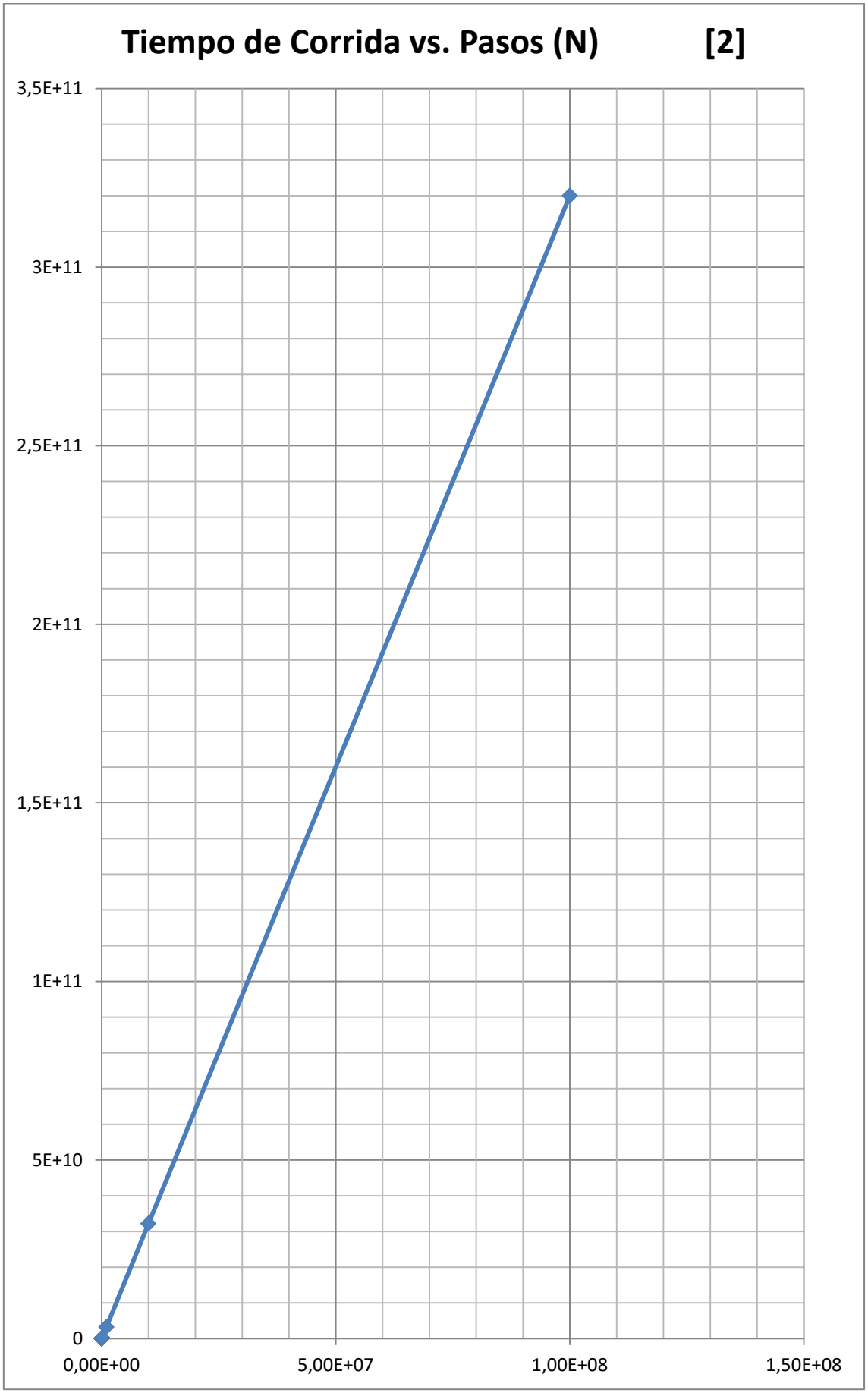
Con q = a (1-e).

Resultados para el algoritmo 1:

K (en radianes)	PASOS (N)	DRR	Tiempo de corrida (en ns)
6,28E-02	1,00E+02	-3,51E-02	0
6,28E-03	1,00E+03	-3,32E-03	0
6,28E-04	1,00E+04	-3,30E-04	0
6,28E-05	1,00E+05	-3,25E-05	0
6,28E-06	1,00E+06	-3,40E-06	3,50E+06
6,28E-07	**1,00E+07**	**-4,17E-07**	**4,00E+07**
6,28E-08	1,00E+08	0	3,10E+08
6,28E-09	1,00E+09	0	3,23E+09
6,28E-10	1,00E+10	0	3,22E+10
6,28E-11	1,00E+11	0	3,20E+11

Continuación - B.4: Gráficos e interpretación




Tiempo de Corrida vs. Pasos (N) [2]
3,5E+11
3E+11
2,5E+11
2E+11
1,5E+11
1E+11
5E+10
0
0,00E+00
5,00E+07
1,00E+08
1,50E+08

Interpretación:

Es importante remarcar la presencia de la nulidad en los últimos resultados para el primer gráfico. Se puede interpretar desde allí que, a partir de las limitaciones que posee el CPU, los errores de redondeos arrastrados luego de cada cálculo producen una cancelación de términos para el cálculo del DRR. Y antes de realizar el análisis de estabilidad del algoritmo, uno podría pensar que esta cancelación podría ser una señal de estar en presencia de un problema mal condicionado.

En varias ocasiones la inestabilidad puede estar dada por la aparición de errores de redondeo, y no por la acumulación de estos. Esto es lo que se busca interpretar en los puntos B.5 y B.8.

En cuanto al segundo gráfico, se puede ver una relación cuasi lineal el tiempo de corrida y los pasos utilizados. En un comienzo el tiempo de cómputo es prácticamente nulo, hasta que se produce un salto brusco y comienza a aumentar el tiempo en orden, tal como lo hace la cantidad de pasos. Debido a que la escala es muy grande se ve una acumulación de puntos, pero en realidad hay una relación entre puntos que es muy similar a la que figura entre los últimos dos puntos del gráfico, se produce siempre una separación que se podría decir es casi periódica, si uno se pusiera a sacar puntos, uno en uno, la relación se ve claramente hasta el momento que llega entre el cuarto y quinto valor que es donde se produjo el "salto" del gráfico.

B.5: Relación entre pasos y DRR. Orden del método.

A medida que los pasos aumentan en orden, el valor de K disminuye más y más, por lo cual se modifica el valor de u_n en cada corrida, causando entonces un aumento cada vez más suave y lento. En relación a la formula dada de DRR, este va a disminuir progresivamente en este sentido. A medida que aumenta N, el valor de DRR podrá ser entonces una mejor aproximación pero hasta cierto caso, porque por ejemplo como ya se vio en los resultados y en los gráficos, se produce una cancelación e términos para un N muy grande, lo que generaría la imposibilidad de representar correctamente, pudiendo significar esto que los errores de redondeo inciden negativamente en el resultado.

Considerando el "orden" del método como la tasa de reducción de DRR con el paso k, se deduce que el "orden" será aproximadamente:

$$\frac{DDR(K_2)/DDR(K_1)}{K_2/K_1}$$

$$\frac{(3{,}40.10^{-6})/(3{,}25.10^{-5})}{(6{,}28.10^{-6})/(6{,}28.10^{-5})} \cong 1{,}05$$

B.6: Análisis de estabilidad por perturbaciones experimentales

Resultados para el algoritmo 1 utilizando doble precisión:

K (en radianes)	PASOS (N)	DRR	Tiempo de corrida (en ns)
6,28E-02	1,00E+02	-3,51E-02	0
6,28E-03	1,00E+03	-3,33E-03	0
6,28E-04	1,00E+04	-3,30E-04	0
6,28E-05	1,00E+05	-3,30E-05	500400
6,28E-06	1,00E+06	-3,30E-06	3,00E+06
6,28E-07	**1,00E+07**	**-3,30E-07**	**3,10E+07**
6,28E-08	1,00E+08	-3,30E-08	3,40E+08
6,28E-09	1,00E+09	-3,30E-09	3,18E+09
6,28E-10	1,00E+10	-3,30E-10	3,20E+10
6,28E-11	1,00E+11	-3,30E-11	3,19E+11

Estimación del factor global de amplificación de errores de redondeo:

$$F_u = \frac{(4{,}17.10^{-7} - 3{,}30.10^{-7})}{4{,}17.10^{-7}.(16 - 8)} \cong 0{,}03$$

Si al analizar un pequeño cambio en los datos (o perturbación) el resultado se modifica levemente (o tiene un pequeño cambio) entonces estamos ante un problema bien condicionado. Si, por el contrario, el resultado se modifica notablemente o se vuelve oscilante, entonces el problema está mal condicionado. Si éste fuera el caso, no hay forma de corregirlo cambiando el algoritmo pues el problema está en el modelo matemático.

B.8: ¿Qué pasa si k < 0.01?

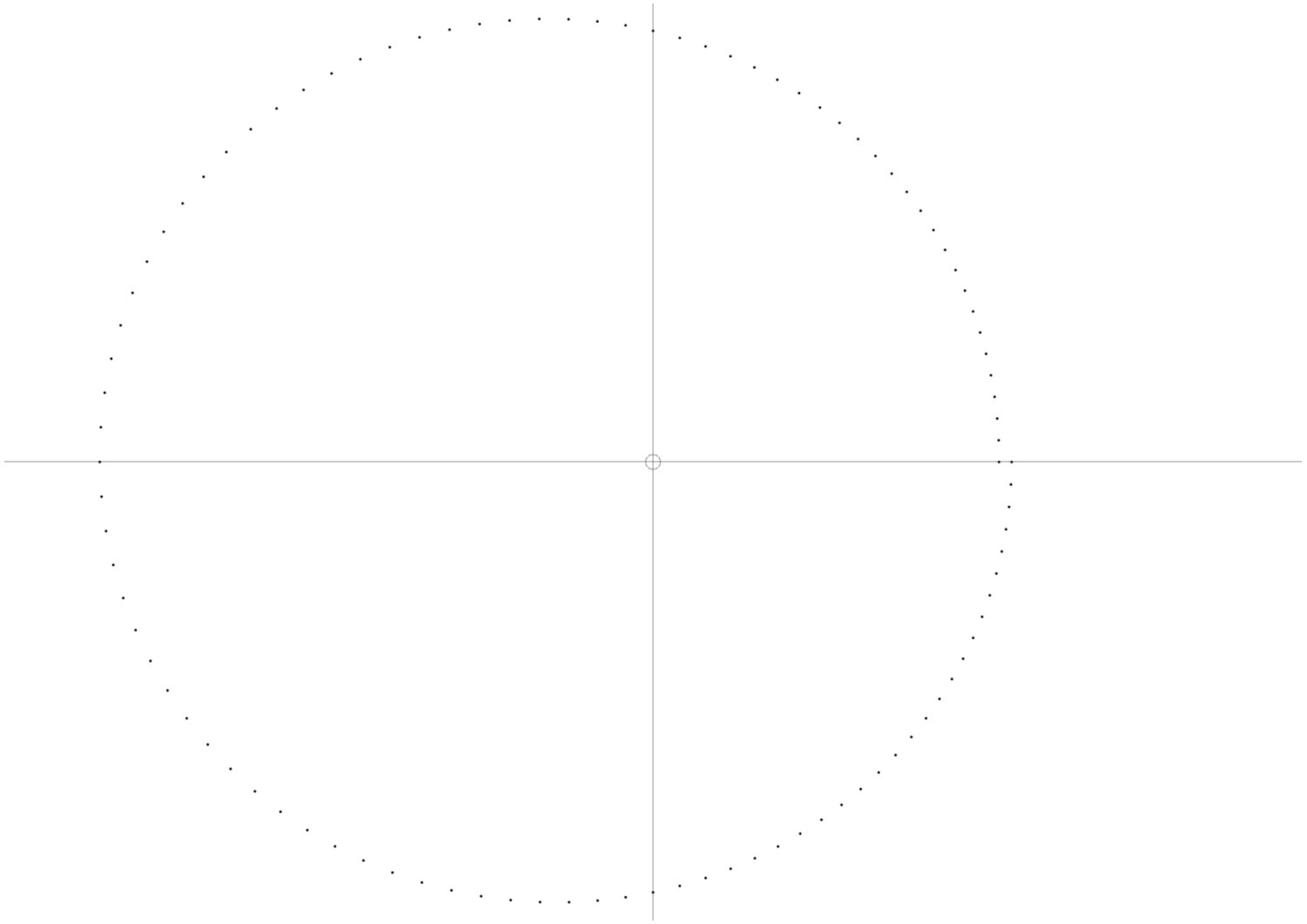
Según los resultados obtenidos por este algoritmo, se puede ver que las soluciones serán confiables siempre y cuando k > 6,28E-02, debido a que por ejemplo, luego de que el paso aumenta un orden más, el k disminuye y el desvío real relativo deja de entregar un valor lógico ya que se vuelve extremadamente pequeño y para las limitaciones que tiene el CPU en este caso el DRR será cero por la aparición de una cancelación de términos. Si se tuviera una CPU con mayor capacidad de cálculo y precisión quizás podrían continuar obteniéndose valores incluso luego de esto, pero tal vez no valga la pena involucrar tanto tiempo y costo de cómputo para poder obtener un resultado más preciso si no es necesario.

Un aumento en la precisión no siempre significa una mejora en los resultados obtenidos. Cuando la única fuente de error es el redondeo (aquí también hay errores de truncamiento), la forma general de corregirlo es aumentar la precisión y comparar los resultados. Pero en el caso de trabajar con un problema mal condicionado el aumento de la precisión no resultará en ninguna mejora.

Si quisiera profundizarse más en esta consigna, significaría repetir conceptos mencionados e interpretados principalmente en los incisos B.5 y B.6, como también B.4.

B.7: Figuras de solución de orbita Mustafar

Para el primer paso (N igual a 100):



Para el ultimo paso (N igual a 100000000):

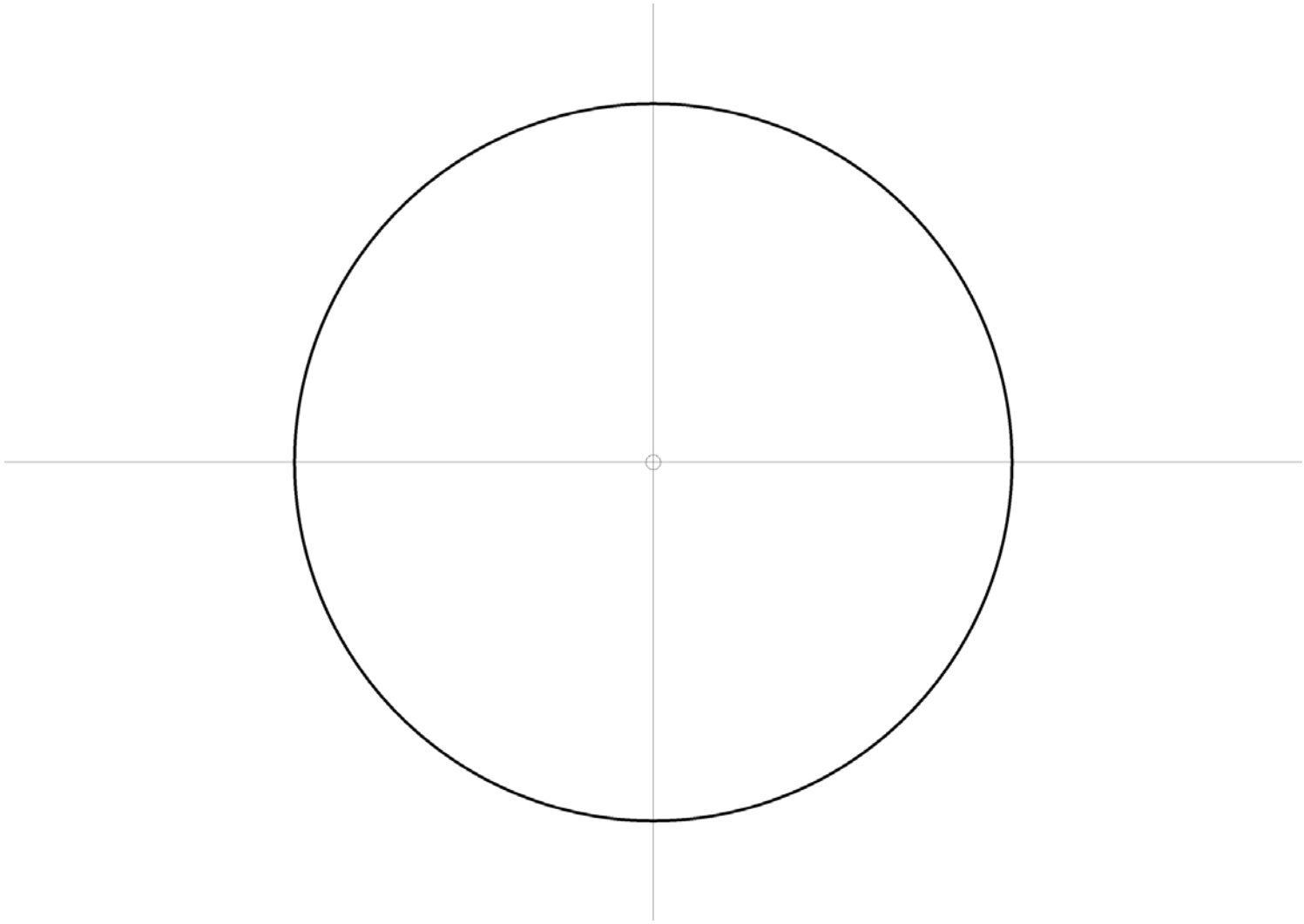
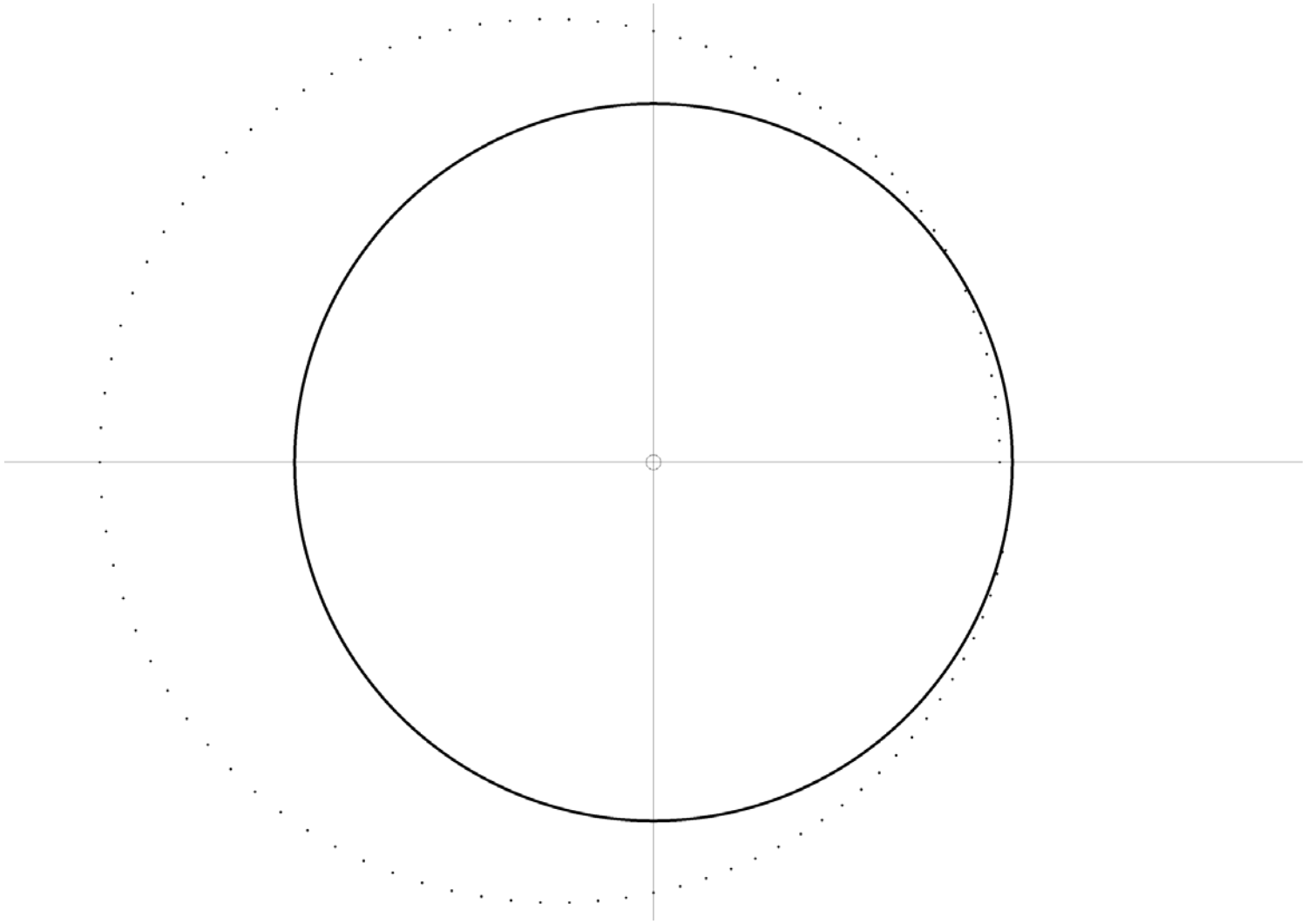


Figura de superposición de ambos gráficos:



2

C. 3 – C.4: Desvío real relativo y tabla de resultados

Por medio de la siguiente formula se pasa a calcular el desvío real relativo (DRR):

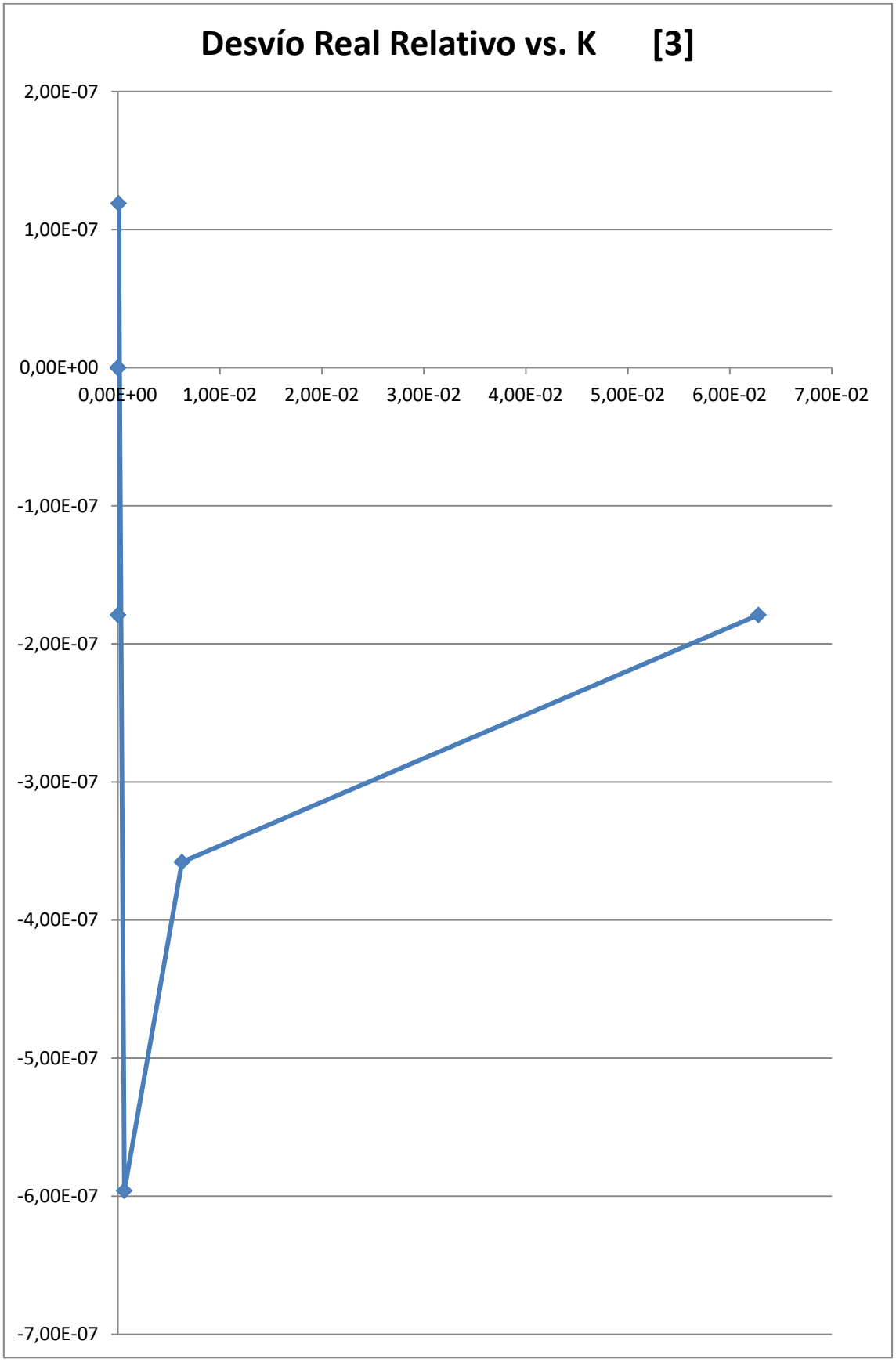
$$\Delta r(N)/\, q = [\, {u_n}^{-1} \,/\, q] - 1$$

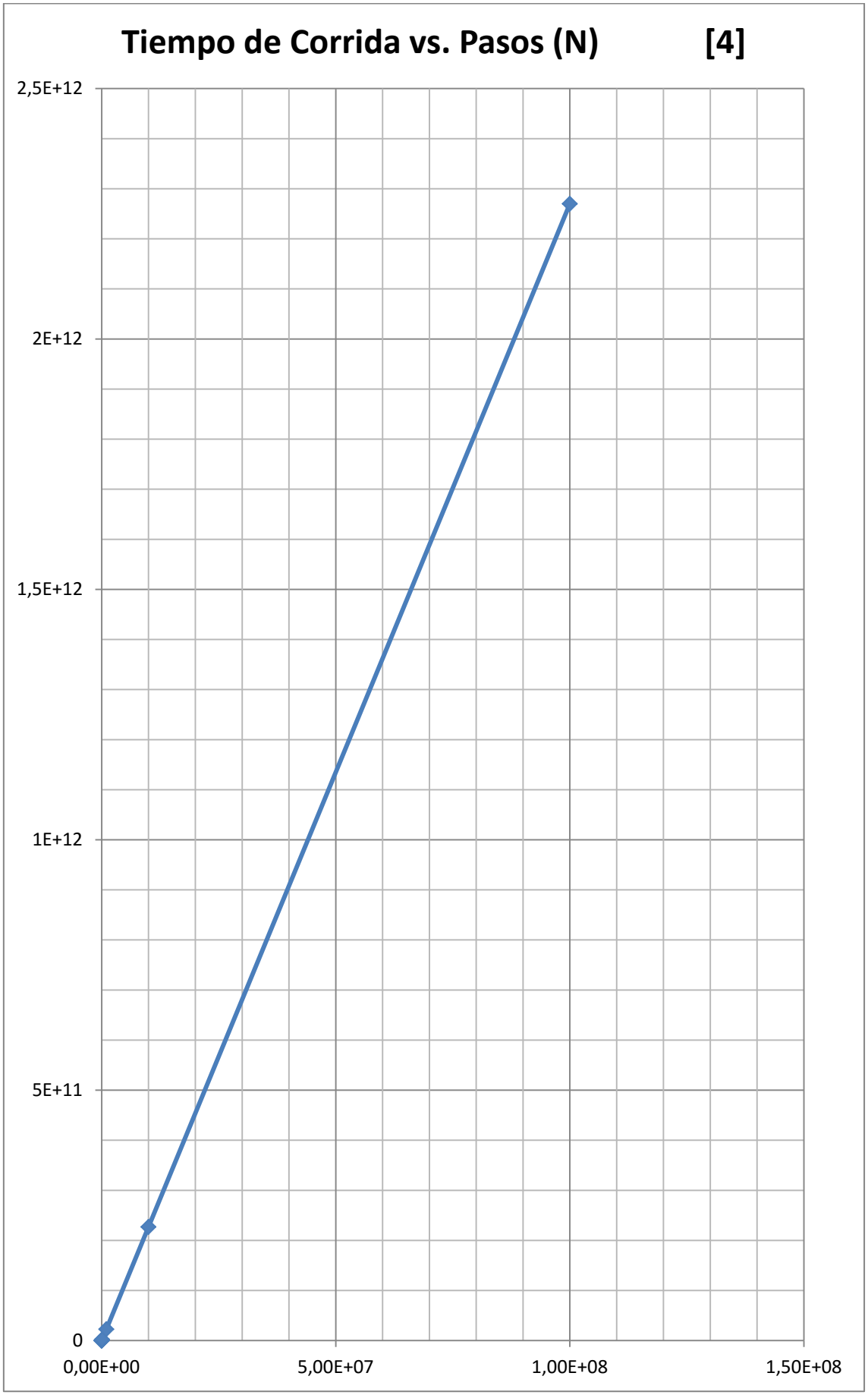
Con q = a (1-e).

Resultados para el algoritmo 2:

K (en radianes)	PASOS (N)	DRR	Tiempo de corrida (en ns)
6,28E-02	1,00E+02	-1,79E-07	0
6,28E-03	1,00E+03	-3,58E-07	0
6,28E-04	1,00E+04	-5,96E-07	0
6,28E-05	1,00E+05	1,19E-07	0
6,28E-06	**1,00E+06**	**-1,79E-07**	**2,98E+07**
6,28E-07	1,00E+07	0	2,20E+08
6,28E-08	1,00E+08	0	2,30E+09
6,28E-09	1,00E+09	0	2,27E+10
6,28E-10	1,00E+10	0	2,27E+11
6,28E-11	1,00E+11	0	2,27E+12

Continuación - C.4: Gráficos e interpretación




Tiempo de Corrida vs. Pasos (N) [4]
2,5E+12
2E+12
1,5E+12
1E+12
5E+11
0
0,00E+00
5,00E+07
1,00E+08
1,50E+08

Interpretación:

En el segundo grafico se produce una relación análoga a la sección B.4, se puede ver una relación cuasi lineal entre el tiempo de corrida y los pasos utilizados, con el mismo tipo de análisis (ver interpretación inciso Continuación B.4), con la diferencia de que el llamado "salto" en esa sección, aquí se produce entre el quinto y sexto valor del gráfico, mencionando que en este caso el tiempo de computo es mayor, sobre todo en los primeros pasos.

Luego, también en relación al punto B.4, este grafico tiene similitud y análisis análogo en relación a la presencia de cancelación de términos, pero hay una diferencia del detalle en cuanto al cálculo del DRR ya que en los primeros términos ahora se ve una variación de valores más espaciado a diferencia del algoritmo uno, lo cual podría significar que el algoritmo dos está mejor condicionado que el uno.

C.5: Relación entre pasos y DRR. Orden del método.

El análisis de la relación entre pasos y DRR es análogo a la sección B.5.

Considerando el "orden" del método como la tasa de reducción de DRR con el paso k, se deduce que el "orden" será aproximadamente:

$$\frac{DDR(K_2)/DDR(K_1)}{K_2/K_1}$$

$$\frac{(1{,}19.10^{-7})/(5{,}96.10^{-7})}{(6{,}28.10^{-5})/(6{,}28.10^{-4})} \cong 2$$

C.6: Análisis de estabilidad por perturbaciones experimentales

Resultados para el algoritmo 2 utilizando doble precisión:

K (en radianes)	PASOS (N)	DRR	Tiempo de corrida (en ns)
6,28E-02	1,00E+02	7,14E-09	0
6,28E-03	1,00E+03	7,11E-14	0
6,28E-04	1,00E+04	-6,66E-16	0
6,28E-05	1,00E+05	-2,66E-15	2,50E+06
6,28E-06	**1,00E+06**	**9,33E-15**	**2,80E+07**
6,28E-07	1,00E+07	-1,55E-15	2,71E+08
6,28E-08	1,00E+08	-1,09E-14	2,67E+09
6,28E-09	1,00E+09	-9,33E-15	2,67E+10

6,28E-10	1,00E+10	-3,36E-14	2,68E+11
6,28E-11	1,00E+11	-3,15E-14	2,69E+12

Estimación del factor global de amplificación de errores de redondeo:

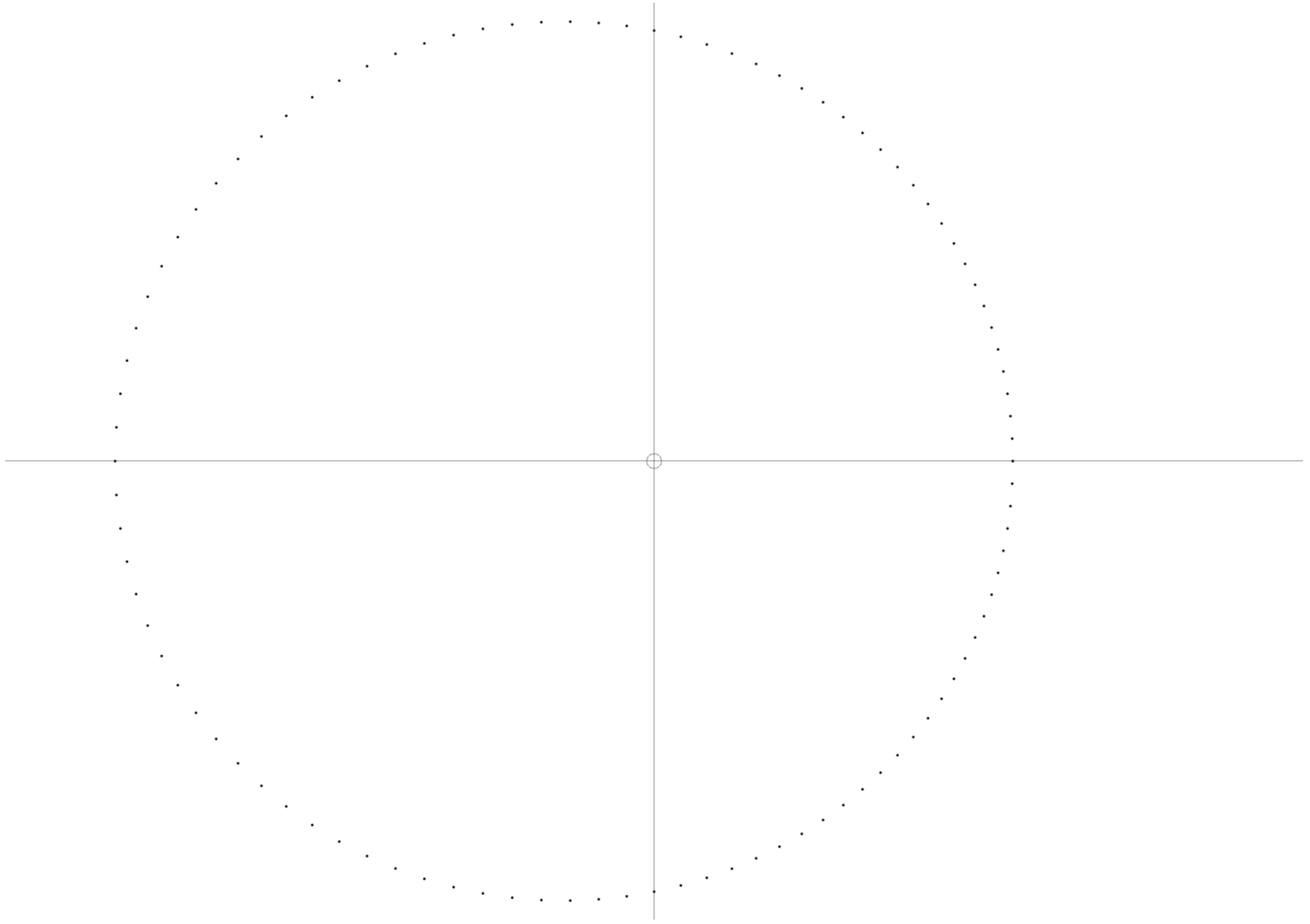
$$F_u = \frac{(9{,}33.10^{-15} - 1{,}79.10^{-7})}{1{,}79.10^{-7}.(16 - 8)} \cong 0{,}12$$

C.8: ¿Qué pasa si k < 0.01?

El análisis de este ítem es análogo al del ítem B.8, salvo que en este algoritmo los resultados serán confiables siempre y cuando k > 6,28E-01, debido a la aparición de una cancelación de términos luego de aumentar el orden (es decir K tiene un orden menor para este caso).

C.7: Figuras de solución de orbita Mustafar

Para el primer paso (N igual a 100):



Para el ultimo paso (N igual a 100000000):

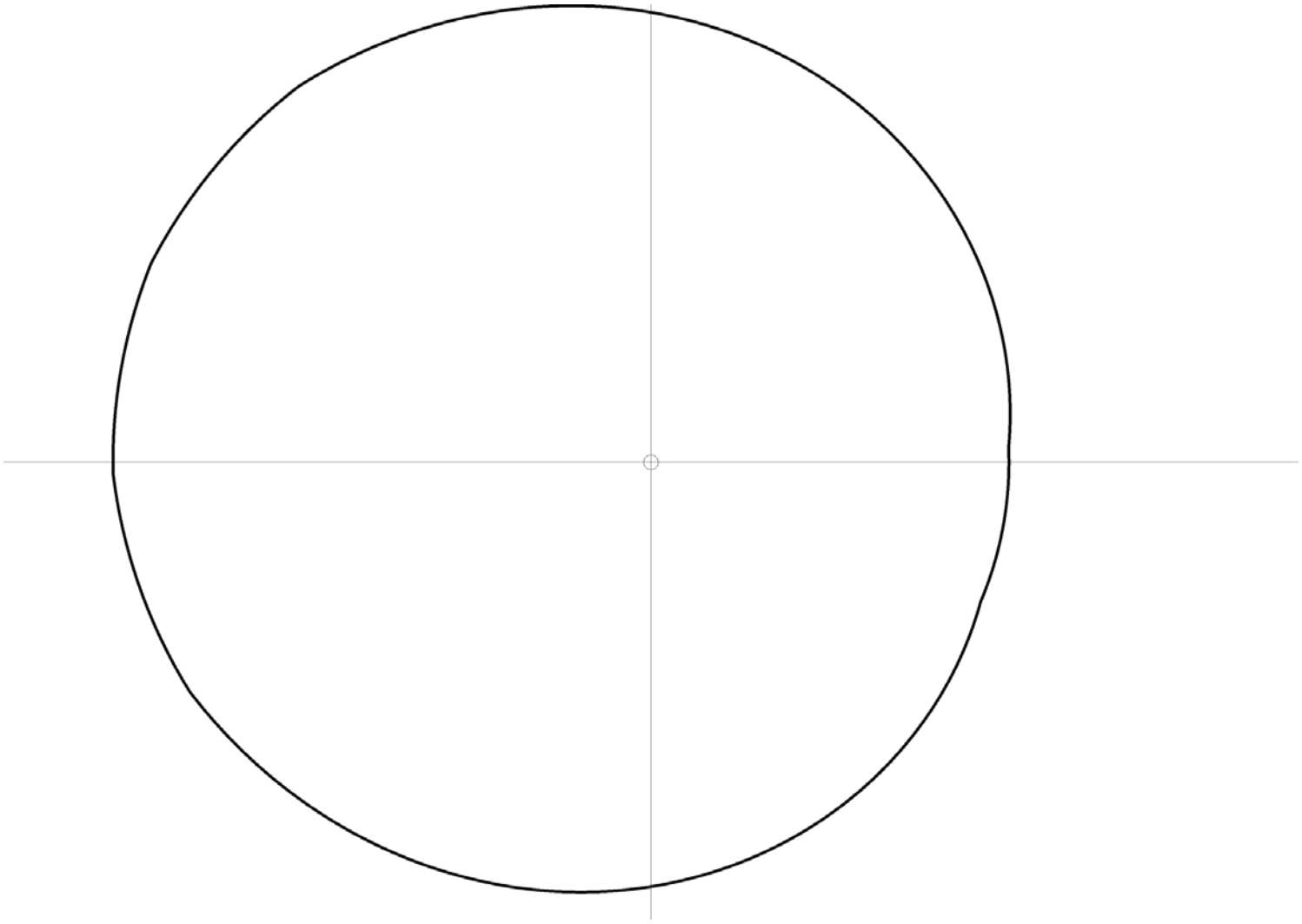
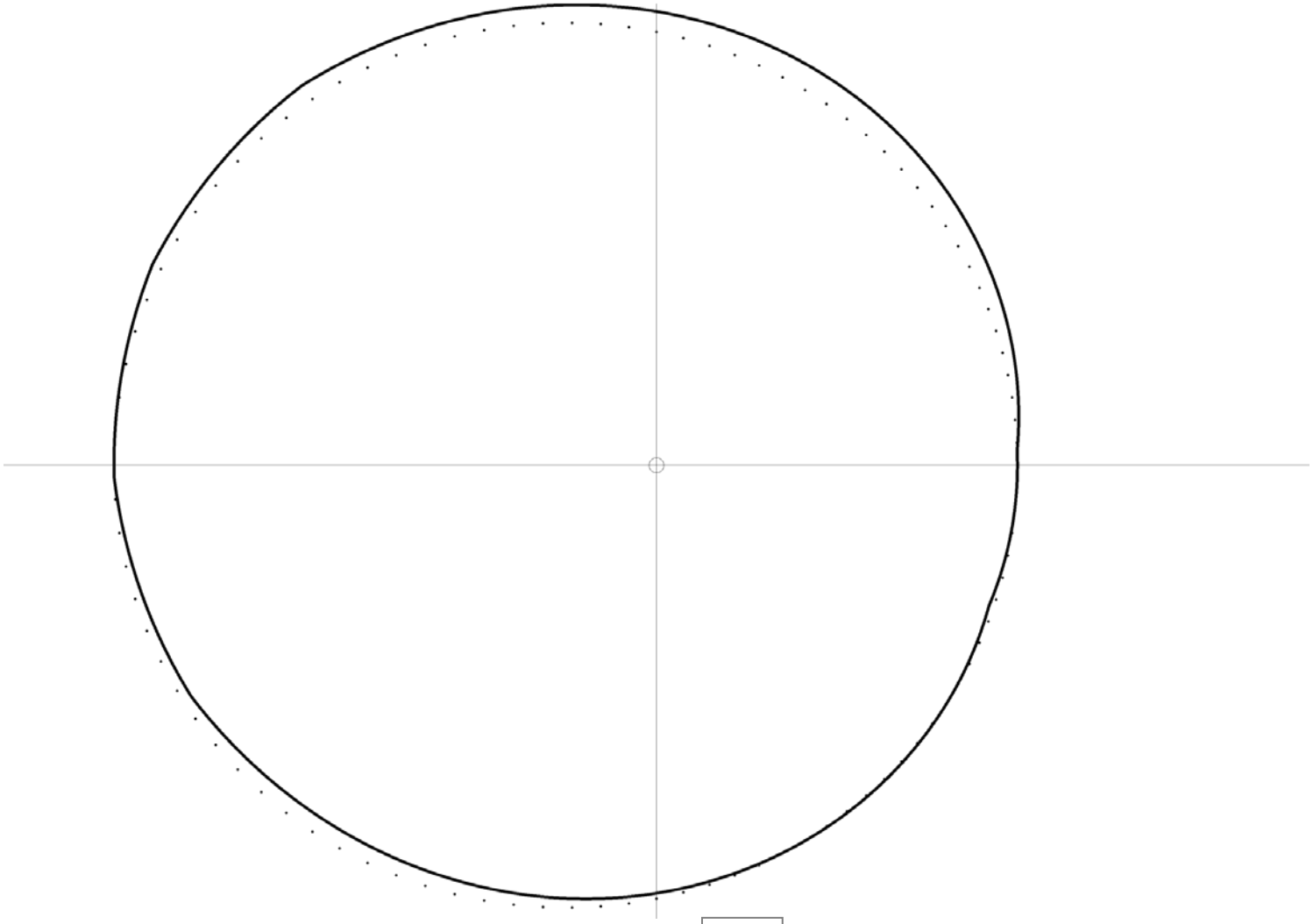


Figura de superposición de ambos gráficos:



Conclusión

El análisis numérico se ocupa de estudiar algoritmos para resolver problemas de la matemática continua. Dado que estos algoritmos son una aproximación al problema matemático, resulta evidente que los resultados obtenidos estarán afectados por algún tipo de error, siendo para este caso el problema matemático el planteado por Newton, y el problema numérico el resuelto por los algoritmos propuestos. Entonces una aproximación se realizará al analizar el la incidencia del *error de redondeo* como también una aproximación será analizar qué ocurre cuando se requiere *truncar* un procedimiento de nuestro problema matemático.

Una de las razones principales de analizar los errores de redondeo es conseguir que un algoritmo sea estable. Dado que la estabilidad es una propiedad exclusiva del algoritmo, si un problema se vuelve inestable podemos, muchas veces, corregirlo cambiando el algoritmo inestable por otro estable. Por esto se realiza el análisis de los algoritmos propuestos, más allá de que hay algunos términos como por ejemplo el número de condición del problema que no depende del algoritmo. En general el análisis numérico se concentra más en estudiar cómo hacer que un algoritmo sea estable más que en analizar su condicionamiento.

A lo largo del análisis del TP se pudo ver principalmente inestabilidad en los algoritmos por medio de la acumulación de errores de redondeo produciendo una cancelación de términos en el DRR, y se podría vislumbrar un peor resultado con respecto al obtenido si se hubiese usado una menor precisión.

Los resultados numéricos obtenidos para estos algoritmos contienen tanto error de redondeo, debido a la precisión finita de la máquina, como error de truncamiento, debido a la naturaleza del problema matemático.

Las principales diferencias y similitudes entre los algoritmos uno y dos son:

- *Velocidad de cómputo*: la del algoritmo dos es mayor que la del algoritmo uno, tomando el máximo tiempo un tiempo de 2,27E+17 nanosegundos para la cantidad de pasos de 1,00E+11
- *Orden*: el orden del algoritmo dos es mayor que la del algoritmo uno.
- *DRR*: mayor propagación de errores para el algoritmo dos ya que se produce la cancelación de términos del DRR antes que la cancelación producida en el algoritmo uno. También el k posee un menor orden al utilizarlo para dar soluciones confiables.
- Relación cuasi-lineal de crecimiento entre N y el tiempo de computo son parecidas para ambos casos.
- Debido a las figuras obtenidas, se ve que el segundo algoritmo está mejor condicionado que el primero, debido a que la solución cuando los pasos son pequeños se parece bastante a la solución con pasos grandes, en cambio para el primero hay una cierta diferencia notable tanto durante la trayectoria como el punto en que finaliza, teniendo siempre en cuenta que nunca, por la propagación de errores que se presenta, llega el grafico al mismo punto desde donde se inició el desplazamiento.

Anexo I: Código fuente

```
/*
**     Universidad de Buenos Aires
**     Facultad de Ingeniería
**     75.12 Análisis Numérico I
**     Trabajo Práctico I
**
**     Merlo Leiva Nahuel -   Fabrizio Luis Cozza
**     Padrón 92115       -   Padrón 97402
*/

#include "stdafx.h"
#include "D2D1Graph.h"

template<typename T>
T computeDDR(T u_n, T semiMinorAxis, T epsilon)
{
	T q = semiMinorAxis * (1 - epsilon);
	T DRR = (pow(u_n, -1) / q) - 1;
	return DRR;
}

template<typename T>
T disturbeSteps(T m1,T m2,T M, T epsilon,T semiMinorAxis) //pasar parametros
ByRef
{
	m1 = lambda * (1.9891e+30) + experimentalDisturbance;; // kg
	m2 = lambda * (3.301e+23) + experimentalDisturbance;; // kg
	M = m1 + m2; // kg
	semiMinorAxis = pow(lambda, 2) * (5.791e+10) + experimentalDisturbance; //
m
	epsilon = pow(lambda, -1) * 0.2056 + experimentalDisturbance; // -
	if (epsilon <= 0 || epsilon >= 1)
	{
		printf("Epsilon=%e out of range!\n", epsilon);
		return 0;
	}
}

template<typename T>
T computeCp(T disturbedResult)
{
	T originalResult = 10; //poner el resultado original aca
	T relativeError = (originalResult - disturbedResult) / originalResult;
	T C_p = relativeError * (1 / experimentalDisturbance); //numero de
condicion del problema
	return C_p;
}

template<typename T>
T computeStability(T resultInOnePrecision, T resultInAnotherPrecision)
{
	//utilizar la precision y los resultados de algun paso en dos precisiones
diferentes y calcular
	machineUnitErrorInAnotherPrecision = 8;
	machineUnitErrorInOnePrecision = 16;
	totalMachineError = machineUnitErrorInAnotherPrecision -
machineUnitErrorInOnePrecision;
	stability = (resultInOnePrecision - resultInAnotherPrecision) /
(resultInOnePrecision * totalMachineError);
```

```
	return stability;
}

template<typename T>
T computeCa(T stability, T C_p)
{
	Ca = stability / C_p;
}

template<typename T>
T algorithm1(T u_0, T v_0, T k, T alpha, double steps, CD2D1Graph* pGraph =
nullptr)
{
	T u_n = u_0;
	T v_n = v_0;
	T k_alpha = k * alpha;
	for (double n = 0; n < steps; n++)
	{
		if (pGraph)
		{
			REAL theta_n = n * k;
			REAL radius_n = 1 / u_n;
			pGraph->DrawPointPolar(radius_n, theta_n);
		}

		T u_n_1 = u_n + (k * v_n);
		T v_n_1 = v_n - (k * u_n) + k_alpha;

		u_n = u_n_1;
		v_n = v_n_1;
	}

	if (pGraph)
	{
		REAL theta_n = steps * k;
		REAL radius_n = 1 / u_n;
		pGraph->DrawPointPolar(radius_n, theta_n);
	}
	return u_n;
}

template<typename T>
T algorithm2(T u_0, T v_0, T k, T alpha, double steps, CD2D1Graph* pGraph =
nullptr)
{
	T u_n = u_0;
	T v_n = v_0;
	for (double n = 0; n < steps; n++)
	{
		if (pGraph)
		{
			REAL theta_n = n * k;
			REAL radius_n = 1 / u_n;
			pGraph->DrawPointPolar(radius_n, theta_n);
		}

		T w_1 = u_n + (k * v_n) / 2;
		T z_1 = v_n + (k * (alpha - u_n)) / 2;
		T w_2 = u_n + (k * z_1) / 2;
		T z_2 = v_n + (k * (alpha - w_1)) / 2;
		T w_3 = u_n + (k * z_2);
		T z_3 = v_n + (k * (alpha - w_2));
```

```
		T u_n_1 = u_n + (k * (v_n + ((2 * z_1) + (2 * z_2) + z_3))) / 6;
		T v_n_1 = v_n + (k * ((6 * alpha) - u_n - (2 * w_1) - (2 * w_2) -
w_3)) / 6;

		u_n = u_n_1;
		v_n = v_n_1;
	}

	if (pGraph)
	{
		REAL theta_n = steps * k;
		REAL radius_n = 1 / u_n;
		pGraph->DrawPointPolar(radius_n, theta_n);
	}
	return u_n;
}

int main(int argc, char* argv[])
{
	double steps = 0;
	long alg = 0;
	long dp = 0;
	long print = 0;
	double scale = 0;
	double width = 0;
	double height = 0;
	for (int argIndex = 1; argIndex < argc; argIndex++)
	{
		if (!strcmp(argv[argIndex], "-alg") &&
			(argIndex + 1) < argc)
		{
			alg = atol(argv[argIndex + 1]);
			argIndex++;
		}
		else if (!strcmp(argv[argIndex], "-steps") &&
			(argIndex + 1) < argc)
		{
			steps = atof(argv[argIndex + 1]);
			argIndex++;
		}
		else if (!strcmp(argv[argIndex], "-dp") &&
			(argIndex + 1) < argc)
		{
			dp = atol(argv[argIndex + 1]);
			argIndex++;
		}
		else if (!strcmp(argv[argIndex], "-print") &&
			(argIndex + 1) < argc)
		{
			print = atol(argv[argIndex + 1]);
			argIndex++;
		}
		else if (!strcmp(argv[argIndex], "-width") &&
			(argIndex + 1) < argc)
		{
			width = atof(argv[argIndex + 1]);
			argIndex++;
		}
		else if (!strcmp(argv[argIndex], "-height") &&
			(argIndex + 1) < argc)
		{
```

```
                height = atof(argv[argIndex + 1]);
                argIndex++;
        }
        else if (!strcmp(argv[argIndex], "-scale") &&
                (argIndex + 1) < argc)
        {
                scale = atof(argv[argIndex + 1]);
                argIndex++;
        }
    }

    if (steps == 0 ||
        alg <= 0
        || alg > 2)
    {
        printf("Wrong params!\n");
        printf("-alg \t Algorithm number {1, 2}.\n");
        printf("-steps \t Algorithm step count.\n");
        printf("-dp \t Use double precision. {0, 1}\n");
        printf("-print \t Print results. {0, 1}\n");
        printf("-scale \t Print scale.\n");
        printf("-width \t Print width.\n");
        printf("-height \t Print height.\n");
        return 0;
    }

    char runDatabaseFileName[255];
    memset(runDatabaseFileName, 0, sizeof(runDatabaseFileName) /
sizeof(runDatabaseFileName[0]));
    sprintf_s(runDatabaseFileName, "alg_%ld_steps_%.0f_dp_%ld_db.csv", alg,
steps, dp);

    printf("Creating output files...\n");

    CD2D1Graph* pGraph = nullptr;
    if (print)
    {
        pGraph = new CD2D1Graph(width, height, scale);
        pGraph->BeginDraw();
    }

    std::fstream statsOutput;
    statsOutput.open("mustafar_stats.csv", std::ios_base::app);

    REAL np = 92115;
    REAL G = 6.673e-11; // N m^2 / kg ^2
    REAL lambda = np / LAMBDA_DEN;
    REAL m1 = lambda * (1.9891e+30); // kg
    REAL m2 = lambda * (3.301e+23); // kg
    REAL M = m1 + m2; // kg
    REAL epsilon = pow(lambda, -1) * 0.2056; // -
    if (epsilon <= 0 || epsilon >= 1)
    {
        printf("Epsilon=%e out of range!\n", epsilon);
        return 0;
    }
    REAL semiMinorAxis = pow(lambda, 2) * (5.791e+10); // m
    REAL mu = G * M; // m^3 / s^2
    REAL specificAngularMomentumSquared = semiMinorAxis * mu * (1 -
pow(epsilon, 2)); // m^4 / s^2
    REAL u_0 = pow(semiMinorAxis * (1 - epsilon), -1);
    REAL v_0 = 0;
```

```
	REAL k = (2 * M_PI) / steps;

	printf("ALGORITHM=%d\n", alg);
	printf("DOUBLE PRECISION=%d\n", dp);
	printf("NP=%e\n", np);
	printf("LAMBDA=%e\n", lambda);
	printf("M1=%e\n", m1);
	printf("M2=%e\n", m2);
	printf("MU=%e\n", mu);
	printf("EPSILON=%e\n", epsilon);
	printf("SEMI MINOR AXIS=%e\n", semiMinorAxis);
	printf("SPECIFIC ANGULAR MOMENTUM SQUARED=%e\n",
specificAngularMomentumSquared);
	printf("U_0=%e\n", u_0);
	printf("V_0=%e\n", v_0);
	printf("STEP COUNT=%e\n", steps);
	printf("RADIAN STEP=%e\n", k);

	statsOutput << alg << ";";
	statsOutput << dp << ";";
	statsOutput << steps << ";";
	statsOutput << (double)k << ";";

	REAL alpha = mu * (pow(specificAngularMomentumSquared, -1));
	REAL u_n = 0;

	printf("Start!\n");
	auto startTimer = std::chrono::high_resolution_clock::now(); //inicio
calculo del tiempo en nanosegundos
	if (alg == 1)
	{
		if (dp)
		{
			u_n = algorithm1<double>(u_0, v_0, k, alpha, steps, pGraph);
		}
		else
		{
			u_n = algorithm1<float>(u_0, v_0, k, alpha, steps, pGraph);
		}
	}
	else if (alg == 2)
	{
		if (dp)
		{
			u_n = algorithm2<double>(u_0, v_0, k, alpha, steps, pGraph);
		}
		else
		{
			u_n = algorithm2<float>(u_0, v_0, k, alpha, steps, pGraph);
		}
	}
	auto finishTimer = std::chrono::high_resolution_clock::now(); //inicio
calculo del tiempo en nanosegundos
	auto tiempoDeCorrida =
std::chrono::duration_cast<std::chrono::nanoseconds>(finishTimer -
startTimer).count();
	printf("End!\n");

	REAL ddr = 0;
	if (dp)
	{
		ddr = computeDDR<double>(u_n, semiMinorAxis, epsilon);
```

```
	}
	else
	{
		ddr = computeDDR<float>(u_n, semiMinorAxis, epsilon);
	}
	printf("DDR=%e\n", ddr);

	printf("Saving stats...\n");
	statsOutput << ddr << ";" << (double)tiempoDeCorrida << ";" << "\n";
	statsOutput.close();

	if (print)
	{
		pGraph->EndDraw();
		pGraph->Present();

		std::wstring pngFileName = _bstr_t(runDatabaseFileName);
		pngFileName.append(L".png");
		pGraph->SavePNG(pngFileName.c_str());
	}

	printf("Done!\n");
	return 0;
}
```

Anexo II: Salida

```
1;0;100;0.0628319;-0.0351377;0;

1;0;1000;0.00628319;-0.00332302;0;

1;0;10000;0.000628319;-0.000330389;0;

1;0;100000;6.28319e-005;-3.25441e-005;0;

1;0;1e+006;6.28319e-006;-3.39746e-006;3.5029e+006;

1;0;1e+007;6.28319e-007;-4.17233e-007;4.00037e+007;

1;0;1e+008;6.28319e-008;0;3.10241e+008;

1;0;1e+009;6.28319e-009;0;3.23389e+009;

1;0;1e+010;6.28319e-010;0;3.21757e+010;

1;0;1e+011;6.28319e-011;0;3.20646e+011;

2;0;100;0.0628319;-1.78814e-007;0;

2;0;1000;0.00628319;-3.57628e-007;0;

2;0;10000;0.000628319;-5.96046e-007;0;

2;0;100000;6.28319e-005;1.19209e-007;0;

2;0;1e+006;6.28319e-006;-1.78814e-007;2.9835e+007;

2;0;1e+007;6.28319e-007;0;2.20347e+008;

2;0;1e+008;6.28319e-008;0;2.2959e+009;

2;0;1e+009;6.28319e-009;0;2.27486e+010;

2;0;1e+010;6.28319e-010;0;2.27258e+011;

2;0;1e+011;6.28319e-011;0;2.27371e+012;


1;1;100;0.0628319;-0.0351377;0;

1;1;1000;0.00628319;-0.00332357;0;

1;1;10000;0.000628319;-0.000330408;0;

1;1;100000;6.28319e-005;-3.30213e-005;500400;

1;1;1e+006;6.28319e-006;-3.30193e-006;3.0016e+006;

1;1;1e+007;6.28319e-007;-3.30192e-007;3.10211e+007;

1;1;1e+008;6.28319e-008;-3.30188e-008;3.40967e+008;

1;1;1e+009;6.28319e-009;-3.30157e-009;3.17764e+009;
```

```
1;1;1e+010;6.28319e-010;-3.30137e-010;3.19733e+010;

1;1;1e+011;6.28319e-011;-3.30146e-011;3.19268e+011;

2;1;100;0.0628319;7.144e-009;0;

2;1;1000;0.00628319;7.10543e-014;0;

2;1;10000;0.000628319;-6.66134e-016;0;

2;1;100000;6.28319e-005;-2.66454e-015;2.5017e+006;

2;1;1e+006;6.28319e-006;9.32587e-015;2.80202e+007;

2;1;1e+007;6.28319e-007;-1.55431e-015;2.70985e+008;

2;1;1e+008;6.28319e-008;-1.08802e-014;2.6738e+009;

2;1;1e+009;6.28319e-009;9.32587e-015;2.67143e+010;

2;1;1e+010;6.28319e-010;-3.36398e-014;2.68259e+011;

2;1;1e+011;6.28319e-011;-3.15303e-014;2.68678e+012;
```