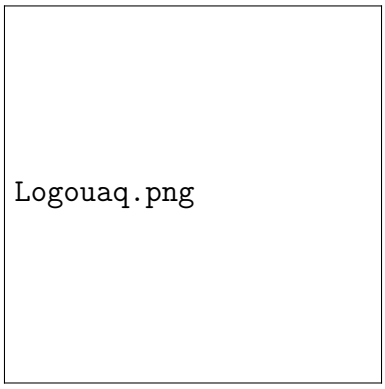
Universidad Autónoma de Querétaro

Facultad de Ingeniería



Tarea 1: Introducción al análisis numérico

Análisis numérico

Autor:
J.A. Salinas Sánchez
Enero 2022

Índice general

Capítulo 1

Introducción

Si has oído hablar, previamente, de que el ingenio humano llega a superar al humano mismo; quiero que sepas que las matemáticas no son una excepción. Pues, esa magnífica forma de poner todo lo existente en este universo en una forma elegante y concisa mediante una notación particularmente simple, generalizable a cualquier objeto, fenómeno, concepto, etc; dado que aquéllo representado no es más que la abstracción más profunda de la vital esencia de lo mismo; en ocasiones se vuelve complicado incluso para nosotros. Y más allá de que lo sea de entender, se torna muy difícil o casi imposible de resolver. Por ejemplo, cualquiera con conocimiento en física y matemáticas avanzadas, puede comprender por qué las ecuaciones de Navier-Stokes modelan todo lo que fluye (no sólo líquidos); no obstante, seguimos sin siquiera poder demostrar que existe una solución exacta en tres dimensiones. Y, si crees que es fácil, te reto. Tú ganas un millón de dólares, y los físicos, una vital pista para descifrar los misterios del universo. Todos ganan.

Si bien, buscar maneras resolver exactamente diferentes cosas de las matemáticas, es interesante, entretenido y trascendental; existe multitud de ocasiones en las que urge obtener un resultado, un número, para resolver nuestros problemas. Es justo ahí donde entra el análisis numérico. El análisis numérico es un conjunto de teorías, algoritmos y técnicas desarrolladas en torno a un punto específico: intentar resolver lo aparentemente irresolvible. Por lo que podría decirse que son todos los "truquitos"que los matemáticos, físicos e informáticos nos han brindado para obtener aproximaciones útiles, en lo que intentan encontrar una manera de domar a esas fieras matemáticas.

Como es de suponerse, en el análisis numérico existe gran y plena variedad de herramientas para la resolución de infinidad de problemas matemáticos: desde aproximar cualquier función, resolver ecuaciones de una o varias variables, hasta resolver ecuaciones diferenciales. Justo por eso sabemos que las ecuaciones de Navier-Stokes sí modelan el comportamiento de los fluídos a pesar de no tenerles solución exacta.

Ahora bien, si estás leyendo un documento introductorio al análisis numérico, supongo que sabes qué son las aproximaciones; si no, lo repito: las aproximar es, mediante un método y razonamiento matemático, obtener un valor cercano a un valor buscado para cualquier propósito. Por ejemplo, cuando, basándote en tu conocimiento sobre el precio de los artículos

del mandado, calculas cuánto dinero podrías llegar a necesitar; sin necesidad de que sea el número exacto.

Luego, como estamos hablando de aproximaciones, es muy probable que el valor obtenido difiera del valor esperado, y dependiendo de cúanto sea esta diferencia tomando en cuenta la precisión y la exactitud necesarias para resolver nuestro problema, se decidirá qué tan útil es el valor obtenido. Aquí es donde entra el análisis de errores. El análisis de errores es, básicamente, utilizar diferentes métodos estadísticos para ver cuánto difieren nuestras aproximaciones y qué tan útiles son según esto.

Para entender el análisis de errores, hay que conocer los tipos de errores:

- Error verdadero: error dado por la comparación entre un valor aproximado y el valor exacto.
- Error aproximado: error dado por la comparación entre dos valores aproximados, pues no se tiene el valor exacto.

De éstos, se desprenden otros, dependiendo de su interpretación estadística:

- Error absoluto: es el valor absoluto de la diferencia entre los valores comparados.
- Error relativo: es el valor del error absoluto en razón del valor dado como verdadero (exacto o aproximado). Nos dice qué tan importante es el error obtenido según nuestro propósito.
- Error normalizado: es el error absoluto en proporción a la incertidumbre de los valores comparados. Nos dice si existe congruencia entre dichos valores.

Finalmente, hay que entender que, cuando se trata de computadoras, dada su naturaleza finita, y a la naturaleza finita del humano, habrá errores en las aproximaciones que hagamos. De ahí surgen los siguientes errores:

- Error de truncamiento: el error que surge por recortar cifras desde el lado del humano.
- Error de redondeo: error que surge por el recorte o redondeo de cifras por parte de la máquina, dadas sus limitaciones.

Hay que tener bien en cuenta que hay que diseñar los algoritmos en función de los errores de redondeo, pues, sin importar cuántas cifras uno quiera manejar desde el lado del programador; si la computadora no puede; no vas a poder.

Capítulo 2

Actividades

```
#ifndef TAREA1_H
#define TAREA1_H


#include <iostream>
#include <cmath>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string>
#include <queue>
#include <stack>

using namespace std;

class Tarea1{
  public:
    static void BinToDec(string & num);
    static long double NumDerivative(long double & num);
    static long double Mepsilon();
    static long double InfSeries();
    static long double RAM();
    static void Polinomio();
  private:
    static long double BinToDecfrac(string & num);
    static long double getnum(long double & num);
    static long double BinToDecint(string & num);
    //static vector <long double> NewtonRaphson(long double &
   prevres, long double & res, vector <long double> & list, int &
   intentos);
    static long double NewtonRaphson();
};

#endif
```

```
#include "Tarea1.h"

void Tarea1 :: BinToDec(string & num){

  bool check = true;

  for(int i = 0;i < num.length();i++){
    if(!isdigit(num[i])){
      check = false;
    }
  }

  if(!check){
```

```
    BinToDecfrac(num);
  }
  else{
    BinToDecint(num);
  }


}

long double Tarea1 :: BinToDecfrac(string & num){

  stack <long double> stfrac;
  stack <long double> stint;
  int i = 0;
  long double dec = 0;

  while(isdigit(num[i])){
    stint.push(num[i]);
    i++;
  }

  i++;

  while(i < num.length()){
    stfrac.push(num[i]);
    i++;
  }

  i = stfrac.size();

  while(!stfrac.empty()){
    dec += getnum(stfrac.top())*pow(2,-1*i);
    i--;
    stfrac.pop();
  }


  while(!stint.empty()){
    dec += getnum(stint.top())*pow(2,i);
    i++;
    stint.pop();
  }

  cout << dec << endl;
  return 0;

}

long double Tarea1 :: BinToDecint(string & num){

  int i;
  stack <long double> stint;
  long double dec = 0;

  for(i = 0;i < num.length();i++){
    stint.push(num[i]);
  }

  for(i = 0;i < stint.size();i++){
    dec += getnum(stint.top())*pow(2,i);
  }

  cout << dec << endl;
```

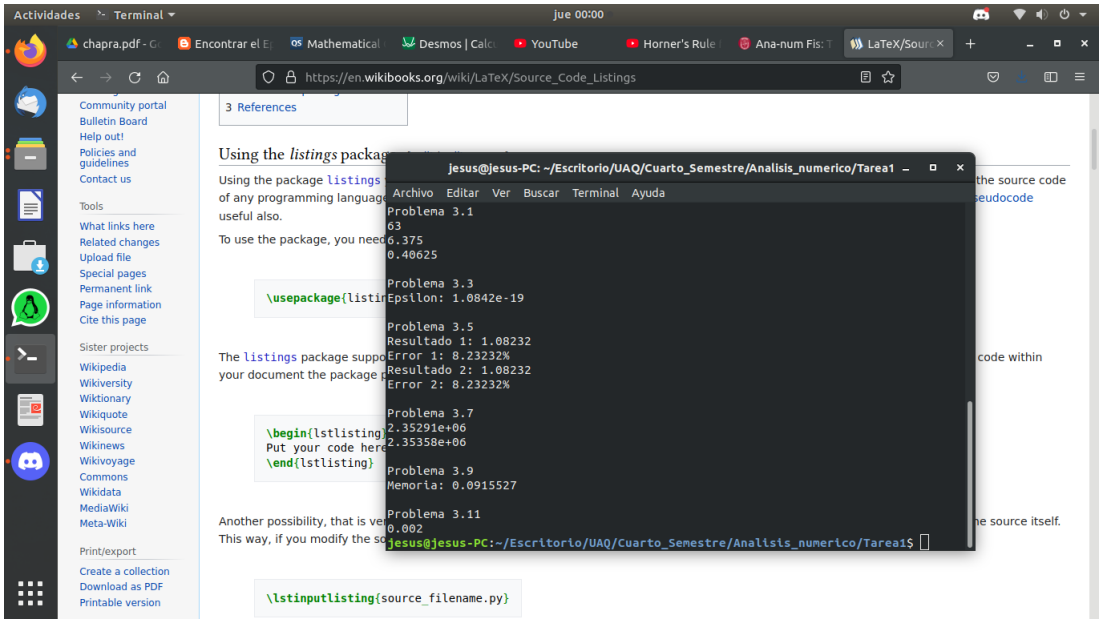
```
  return 0;
}

long double Tarea1 :: Mepsilon(){

  long double epsilon = 1;
        long double prevep;

        while(epsilon+1 != 1){
                prevep = epsilon;
                epsilon /= 2;
        }

        cout << "Epsilon: " << prevep << endl;
        return 0;

}


long double Tarea1 :: InfSeries(){

  int i;
  int j=10000;
  long double s1 = 0;
  long double s2 = 0;

  for(i = 1;i <= 10000;i++){
    s1+=1/(pow(i,4));
    s2+=1/(pow(j,4));
    j--;
  }

  int realvalue = pow(M_PI,4)/90;

  cout <<  "Resultado 1: " << s1 << endl;
  cout << "Error 1: " << (abs(realvalue-s1)/realvalue)*100 << "%"
   << endl;
  cout <<  "Resultado 2: " << s2 << endl;
  cout << "Error 2: " << (abs(realvalue-s2)/realvalue)*100 << "%"
   << endl;

  return 0;
}

long double Tarea1 :: NumDerivative(long double & num){

  long double df=6*num/(pow(1-3*pow(num,2),2));

  cout << df << endl;

  long double Df=((1/(1-3*pow(num+0.0000001,2)))-(1/(1-3*pow(num,2)
   )))/(0.0000001);

  cout << Df << endl;
  return 0;

}

long double Tarea1 :: RAM(){

  long double volume = 96000;
  long double memory = volume/1048576;

```

```
  cout << "Memoria: " << memory << endl;
  return 0;
}

void Tarea1 :: Polinomio(){
  NewtonRaphson();
  /*long double a = 1;
  long double b = 1;
  int c = 0;
  vector <long double> v = {};
  NewtonRaphson(a,b,v,c);
  for(int i = 0; i < v.size(); i++){
    cout << v[i] << endl;
  }*/
}

/*vector <long double> Tarea1 :: NewtonRaphson(long double &
   prevres, long double & res, vector <long double> & list, int &
   intentos){
  int grado = 2;
  int i;
  bool found = false;
  if(list.size() == grado || intentos >= grado+1){
    return list;
  }
  else if(abs(prevres-res)<0.00000001){
    for(i = 0;i < list.size();i++){
      if(res == list[i]+0.00000001 || res == list[i]-0.00000001){
        found = true;
        break;
      }
    }
    if(found){
      intentos++;
      NewtonRaphson(prevres,res,list,intentos);
    }
    else{
      list.push_back(res);
      if(prevres-res < 0){
        prevres = res;
        res += res;
      }
      else if(prevres-res > 0){
        prevres = res;
        res -= res;
      }
      intentos++;
    }

  }

  res = prevres - (pow(prevres,2)-5000.002*prevres+10)/(2*prevres
   -5000.002);
  prevres = res;

  res = prevres - (pow(prevres,2)-5000.002*prevres+10)/(2*prevres
   -5000.002);
  NewtonRaphson(prevres,res,list,intentos);

}*/

long double Tarea1 :: NewtonRaphson(){
  long double prevres = 1;
```

```
  long double res = 0;
  int intentos = 0;

  while(abs(prevres - res) > 0.001){
    res = prevres - (pow(prevres,2)-5000.002*prevres+10)/(2*prevres
    -5000.002);
          prevres = res;

          res = prevres - (pow(prevres,2)-5000.002*prevres+10)/(2*
    prevres-5000.002);
  }
  cout << res << endl;


}

long double Tarea1 :: getnum(long double & num){
  if(num == 48){
    return 0;
  }
  else if(num == 49){
    return 1;
  }
}
```

```
#include "Tarea1.h"


int main(){

  cout << "" << endl;
  cout << "Problema 3.1" << endl;

  string a = "101101";
  Tarea1 :: BinToDec(a);
  a = "110.011";
  Tarea1 :: BinToDec(a);
        a ="0.01101";
  Tarea1 :: BinToDec(a);

  cout << "" << endl;
  cout << "Problema 3.3" << endl;
  Tarea1 :: Mepsilon();

  cout << "" << endl;
  cout << "Problema 3.5" << endl;
  Tarea1 :: InfSeries();

  cout << "" << endl;
  cout << "Problema 3.7" << endl;
  long double b = 0.577;
  Tarea1 :: NumDerivative(b);

  cout << "" << endl;
  cout << "Problema 3.9" << endl;
  Tarea1 :: RAM();

  cout << "" << endl;
  cout << "Problema 3.11" << endl;
  Tarea1 :: Polinomio();

  return 0;

```

```
39 }
```


Actividades
Terminal
jue 00:00
chapra.pdf - G
Encontrar el E
Mathematical
Desmos | Calc
YouTube
Horner's Rule
Ana-num Fis: T
LaTeX/Sourc
https://en.wikibooks.org/wiki/LaTeX/Source_Code_Listings
Community portal
Bulletin Board
Help out!
Policies and guidelines
Contact us
Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Cite this page
Sister projects
Wikipedia
Wikiversity
Wiktionary
Wikiquote
Wikisource
Wikinews
Wikivoyage
Commons
Wikidata
MediaWiki
Meta-Wiki
Print/export
Create a collection
Download as PDF
Printable version
3 References
Using the listings packag
Using the package listings
of any programming languag
useful also.
To use the package, you need
\usepackage{listi
The listings package suppo
your document the package
\begin{lstlisting}
Put your code her
\end{lstlisting}
Another possibility, that is ve
This way, if you modify the s
\lstinputlisting{source_filename.py}
the source code
seudocode
code within
he source itself.
jesus@jesus-PC: ~/Escritorio/UAQ/Cuarto_Semestre/Analisis_numerico/Tarea1
Archivo Editar Ver Buscar Terminal Ayuda
Problema 3.1
63
6.375
0.40625
Problema 3.3
Epsilon: 1.0842e-19
Problema 3.5
Resultado 1: 1.08232
Error 1: 8.23232%
Resultado 2: 1.08232
Error 2: 8.23232%
Problema 3.7
2.35291e+06
2.35358e+06
Problema 3.9
Memoria: 0.0915527
Problema 3.11
0.002
jesus@jesus-PC:~/Escritorio/UAQ/Cuarto_Semestre/Analisis_numerico/Tarea1$