Interactive-Numeric Documentation

Release 1.2

Semillero de Fisica Teorica y Computacional

March 16, 2015

ONE

CAPITULO 00: INTRODUCCIÓN

Esta es la documentación oficial del proyecto cofinanciado por Colciencias desarrollado por el Semillero de Física Teórica y Computacional https://sites.google.com/site/fisicatyc/home.

El proyecto Numéricos - Interactivos es una implementación en Python 3.4 de los métodos numéricos elementales de Búsqueda de Raíces, Interpolación, Diferenciación e Integración.

El repositorio vigente del proyecto puede encontrarse en GitHub en el vínculo:

https://github.com/fisicatyc/numericos-interactivo

Requerimientos:

> El proyecto fue desarrollado utilizando las funciones Built-In de Python 3.4 por lo que no es necesario otro tipo paquetes para su funcionamiento.

Contenidos:

MÉTODOS DE BÚSQUEDA DE RAÍCES

2.1 Métodos Cerrados

Los métodos cerrados son métodos que requieren de dos valores iniciales para la búsqueda de raíces, los cuales son los valores de los extremos de un intervalo que encierra una raíz. Estos métodos siempre son convergentes pero suelen ser algo lentos.

En los métodos cerrados la metodología se puede describir como una sucesión de pasos que cada vez acotan más el lugar de la raíz, es decir, disminuyen el tamaño del intervalo. Para poder asegurar que una raíz se halla al interior de este intervalo debemos cumplir con que:

$$f(a)f(b) \leq 0 \tag{2.1}$$

Siendo a y b los extremos del intervalo actual y f una función continua en el intervalo cerrado $[a,b]$. Si esta condición se cumple tendremos:

$$\exists c \in [a,b] \, / \, f(c) = 0$$

Este resultado se obtiene a partir del *teorema de Bolzano*. Dado que la función es continua recorrerá todos los valores comprendidos entre *f(a)* y *f(b)*, y como existe cambio de signo entre sus extremos uno de estos valores para la función será igual a cero, por ende, entre a y b existe una raíz.

Nótese que si la función presenta raíces de multiplicidad par (un numero par de raíces idénticas) los métodos cerrados no convergen.

$$\left(\lim_{a \to c^-} f(a) \right) \left(\lim_{b \to c^+} f(b) \right) > 0$$

La diferencia entre los métodos cerrados se basa en la forma de reducir el intervalo alrededor del cual se halla la raíz, o en otras palabras la aproximación del valor de la raíz. De ahí métodos como búsquedas incrementales, bisección y regla falsa.

2.1.1 Búsquedas Incrementales

Los métodos de búsquedas incrementales aprovechan la característica de la ec.2.1 para determinar el intervalo en el cuál se encuentra por lo menos una raíz, con un error igual a la mitad del tamaño de paso del avance en el intervalo. Para mejorar la precisión del método, una vez localizado el intervalo que presenta cambio de signo, este se puede subdividir en mas intervalos y nueva mente aplicar el método de búsquedas incrementales.

La formulación iterativa y el error absoluto son expresados a continuación, siendo n el número de iteraciones máxima o puntos totales ($n - 1$ subintervalos explorados) e i el i-ésimo subintervalo (o iteración). a y b son los extremos del intervalo y c la aproximación de la raíz.

$$c = a + i\frac{b - a}{n}$$
$$\epsilon = \frac{b - a}{2(n - 1)}$$

(2.2)

2.1.2 Bisección

El método de bisección es un tipo de búsqueda incremental en el que el intervalo siempre se divide en 2 subintervalos. n es el número de iteraciones máxima (número de veces en que se subdividirá el intervalo).

$$c = \frac{a + b}{2}$$
$$\epsilon = \frac{b - a}{2^{n+1}}$$

(2.3)

Este método al igual que búsquedas incrementales, solo realizan consideraciones del intervalo mas no del comportamiento de la función en el intervalo.

2.1.3 Regla Falsa

Este método introduce a la aproximación de la raíz un conocimiento del comportamiento de la función. El hecho que un valor de la función este próximo a cero puede indicar igualmente que la posición de la variable independiente este cerca de la raíz real. Para determinar la aproximación de la raíz se unen los puntos de los extremos por una línea recta, y la intersección de esta con el eje x constituye una muy buena aproximación de la raíz.

La obtención de formulación iterativa se obtiene a partir de triángulos semejantes.

$$\frac{f(a)}{c - a} = \frac{f(b)}{c - b}$$

(2.4)

Dada la consideración del comportamiento de la función tenida en regla falsa, este método en general presenta una convergencia mayor a los otros métodos cerrados. En algunos casos, el método presenta puntos de estancamiento y debe plantearse un criterio de parada adicional al error y modificar la forma iterativa del método alterando la magnitud de uno de los extremos del intervalo.

2.2 Métodos Abiertos

Los métodos abiertos son métodos cuya formulación solo requiere de un valor inicial para estimar la raíz de la función. Estos métodos no necesariamente son convergentes, pero cuando lo logran generalmente son mas rápidos que los métodos cerrados. Esta situación puede darse gracias a que los métodos abiertos toman consideraciones acerca de la geometría de la función particular que se desea.

La forma general de estos métodos es dada por la expresión ec.2.5, donde a partir de una ecuación $f(x) = 0$ se obtiene:

$$x = g(x) \tag{2.5}$$

Para que el método sea convergente se debe cumplir que en ec.2.5

$$\left| g\prime(x) \right| < 1 \tag{2.6}$$

donde g es una función tal que $g\colon \mathbb{R} \to \mathbb{R}$

2.2.1 Método de Punto Fijo

La iteración de punto fijo exige llevar la ecuación de la forma $f(x) = 0$ a la forma ec.2.5. Una vez hecho esto, se realiza una sustitución sucesiva, teniendo la forma iterativa.

$$x_{i+1} = g(x_i) \tag{2.7}$$

En la cual la función g cumple con ec.2.6. El método de punto fijo posee convergencia lineal.

Ejemplo 2.1. Sea $f(x) = \exp^{-x} - x$, obtener la iteración de punto fijo.

$$e^{-x} - x = 0$$
$$x_{i+1} = e^{-x_i}$$

Ejemplo 2.2. Sea $f(x) = x^2 - 2x + 3$, obtener la iteración de punto fijo.

$$x^2 - 2x + 3 = 0$$
$$x_{i+1} = \frac{x_i^2 + 3}{2}$$

2.2.2 Método de Newton-Raphson

Dada una función $f(x)$ su expansión en series de Taylor alrededor de un punto x_0 puede ser expresada como:

$$f(x) = f(x_0) \left. \frac{df(x)}{dx} \right|_{x=x_0} (x - x_0) + \frac{1}{2} \left. \frac{d^2 f(x)}{dx^2} \right|_{x=\xi} (x - x_0)^2 \tag{2.8}$$

Tomando el orden 1 de aproximación en ec.2.8 y despejando para x:

$$x = \frac{f(x) - f(x_0)}{f\prime(x_0)} + x_0$$

finalmente, nos interesa la raíz de la función por lo que $f(x) = 0$.

$$x_{i+1} = x_i - \frac{f(x_i)}{f\prime(x_i)} \tag{2.9}$$

El método de Newton, cuya iteración es descrita por la ec.2.9 posee una convergencia cuadrática. El método de Newton puede tener modificaciones útiles para el caso de raíces múltiples. Así siendo m la multiplicidad puede definirse la iteración como:

$$x_{i+1} = x_i - m \frac{f(x_i)}{f\prime(x_i)} \tag{2.10}$$

Otra variación del método de Newton consiste en redefinir la función a la cual se buscaran las raíces. El método tiene la forma general:

$$x_{i+1} = x_i - \frac{u(x_i)}{u\prime(x_i)} \tag{2.11}$$

con $u(x) = f(x)$. Debemos proponer una función $u(x)$ tal que sus raíces sean las mismas que $f(x)$.
Si $f\prime(x) \neq 0$ las raíces de

$$u(x) = \frac{f(x)}{f\prime(x)}$$

serán las mismas de $f(x)$. Sustituyendo esta en ec.2.11 se obtiene la formulación de Newton modificada para raíces múltiples.

2.2.3 Método de Secante

El método de secante consiste en una alternativa al método de Newton para el calculo de la derivada de la función. Esto es ya que no siempre será posible calcular esta o dicho calculo puede constituye un costo computacional mayor, por lo que la expresión de ec.2.9 se modifica incorporando una forma discreta de la derivada.

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)} \tag{2.12}$$

Es posible considerar una variante del método de secante en la cual para la aproximación de la derivada se considera un pequeño cambio en la variable independiente de la función:

$$x_{i+1} = x_i - \frac{f(x_i)\Delta x}{f(x_i + \Delta x) - f(x_i)} \tag{2.13}$$

2.3 Implementación del método:

En este módulo se aplican los métodos de búsqueda de la raíces de una función, lo cual vendría a ser la búsqueda de todo elemento x que pertenezca al dominio de la función y que dicha función cumpla f(x) = 0.

Se pretende que el usuario aplique los métodos a la solución de problemas, para lograr una mejor compresión sobre la temática y se adquieran habilidades que faciliten el progreso hacia otros métodos de mayor complejidad.

Dentro del paquete rootFinders se encuentran los siguientes métodos para la búsqueda de raíces en una función:

-Bisection Method -FixedPoint Method -Müller Method -PolyRoot Method -RegulaFalse Method -Secant Method

2.4 Secuencia de llamado:

Cada método corresponde a un archivo de extensión .py, que en el paquete se pueden encontrar como:

Note: bisection.py, fixedpoint.py, muller.py, polyroot.py, regulafalsi.py, secant.py.

Para ejecutar éstas funciones desde cualquier terminal de Python 3.4, es necesario cargar en memoria el paquete, con la línea:

Note: from Numericos.lib.rootFinders import *

Éste comando cargará simultaneamente los seis métodos del paquete. Para ejecutar un cálculo, es necesario agregar 'rootFinders.(método)', y sus respectivos argumentos. En el caso de bisección la sequencia sería:

Note: rootFinders.bisection(('6*x-6'),-5,6,200,0.005);

En el caso de cargar un único método, se utiliza el comando:

Note: from Numericos.lib.rootFinders import muller

De ésta manera solamente el método de Müller esta disponible para operar. Ésta forma simplifica la forma de llamado.

CAPITULO 02: INTERPOLADORES

En general, para las aplicaciones de la ingeniería y de la investigación aplicada, sera necesario partir de información obtenida experimentalmente, lo que traduce en que la información que poseemos no es una función continua que describe el fenómeno sino de un conjunto de puntos cuya única relación inicial es que fueron obtenidos de un mismo experimento u observación de un fenómeno. El paso siguiente a partir de la obtención de estos puntos, es la obtención de una curva que describa adecuadamente los puntos (ya sea aproximándolos -curvas de regresión- o atravesándolos -curvas interpolantes-) y a partir de las cuales sea posible construir modelos matemáticos para la descripción del fenómeno y permitan realizar predicciones acerca del mismo (al igual que estudiar el comportamiento para las condiciones no obtenidas experimentalmente). En cuanto a la interpolación, se recurre generalmente a la interpolación polinomial.

Para un conjunto de $n + 1$ puntos, existe un ubico polinomio de grado n que pasa por todos los puntos. Para construir este polinomio existen varias técnicas, como lo son Interpolación de Newton en diferencias divididas, Interpolación de Lagrange, calculo de los coeficientes directamente mediante un sistema de ecuaciones y otros. Igualmente, existen métodos de interpolación, que producen curvas suaves que pasan por todos los puntos, y que en general pueden acercarse mucho mas a la descripción del fenómeno, mas no son un polinomio, sino funciones por tramos que se constituyen de polinomios de grados bajos. Estos son los splines o trazadores.

3.1 Interpolación de Newton

Se puede formular de manera general como

3.2 Descripción del método:

Este módulo se encarga de la implementación de los métodos de interpolación, que se basa en la problemática de tener n+1 puntos secuenciales, donde x0 < x1 < x2 < x3 < ... < xn, y a partir de estos puntos diferentes se va a definir una función polinómica que pase por esos puntos. Básicamente se está aproximando p(x)= y a una f(x)= y original, a la cual probablemente no se tiene acceso o conocimiento. Esto se hace con el fin de modelar el comportamiento de los puntos anteriores y obtener el "polinomio interpolante".

Se encuentran los siguientes métodos interpoladores:

-Beziercurve Method -Clampspline Method -Derivpol Method -Hermitpoly Method -Nevilletable Method -Newtondivdef Method -Natspline Method

CAPITULO 03: DIFERENCIADORES

4.1 Descripción del método:

En este módulo se aplican los métodos diferenciadores o también conocidos como diferencias finitas. Se van a categorizar como diferencias centradas y no centradas, donde la primera define un conjunto de métodos numéricos en diferencias finitas con paso centrado. Y en los métodos no centrados se define un conjunto de métodos numéricos basándose en diferencias finitas SIN paso centrado. Estas técnicas se emplean muy a menudo en análisis numérico, especialmente en ecuaciones numéricas ordinarias, ecuaciones en diferencias y ecuaciones En derivadas parciales. Las aplicaciones habituales suelen ser en los campos de la computación, áreas de la ingeniería térmica o en la Mecánica de fluidos.

Se encuentran los siguientes métodos diferenciadores centrados:

-QCDA or Fourth Centered Difference Aproximation, calculates the fourth derivative in the interval.

-SCDA or Second Centered Difference Aproximation, calculates the second derivative in the interval.

-TCDA or Third Centered Difference Aproximation, calculates the third derivative in the interval.

-QCDA or Fourth Centered Difference Aproximation, calculates the fourth derivative in the interval.

Se encuentran los siguientes métodos diferenciadores no centrados:

-nfirstderv -nsecondderv -snfirstderv

FIVE

CAPITULO 04: INTEGRADORES

5.1 Descripción del método:

En este módulo se encuentran los métodos integradores que aproximan la funcion f(x) que se va a integrar por medio de polinomios interpolantes de diferentes grados, donde los métodos varían según el número de puntos o intervalo que se utilice para dicha aproximación permitiendo calcular integrales definidas. Existen varios métodos que permiten realizar el proceso de hallar una integral defina y se dividen en este proyecto en las siguientes categorías:

        -ClosedNewtonCotes   *Fournodescotes *simprule *tesimprule *trapzrule

        -CompositeRules   *Compmidtrule *compsimrule *comptraprule

        -DoubleIntegral   *Gaussblint *Simpbdlintg

        -OpenNewtonCotes *Midpointrule *Onenodeopn *Thrnodeopn *twonodeopn

        -Tripleintegral   *Gausstpllint

   -Adapquad -Natspline -Romberginted

CAPITULO 05: EXTRAPOLADORES

6.1 Descripción del método:

Este módulo se encarga de los métodos de extrapolación, son métodos científico lógicos, que se basan en suponer que el curso de los acontecimientos continuara, es decir, afirma que existen unos axiomas y que estos son extrapolables a una nueva situación. La base de la extrapolación son observaciones secuenciales realizados en periodos conocidos de tiempo, estas observaciones son luego registradas como variables cuantitativas, medidas con algún tipo de escala.

Se usan para buscar soluciones a problemas lógicos o enseñar la misma pedagogía, lo cual los convierte en una herramienta muy utilizada en el marco profesional y de enseñanza, no son necesariamente exclusivos de los métodos de interpolación y tampoco se pueden considerar únicos.

En el marco de los métodos numéricos, los extrapoladores permiten recalcular, a partir de dos aproximaciones, un valor más preciso. Ésta herramienta es la piedra angular de los métodos de cuadratura. Al estar integrado en un método particular, los valores que arroja cada dos iteraciones son refinados. En la próxima iteración se obtiene una nueva aproximación que se refina con el valor extrapolado generando un valor aún más preciso. Se cuenta con los siguientes métodos:

-Richarexpol o Extrapolación de Richardson

SEVEN

DOCUMENTACIÓN E ÍNDICE DE LOS MÉTODOS - ENGLISH VER.

7.1 adapquad.py

Description: function adapquad is an adaptive quadrature method based on Simpson's rule for integration. It can also can be expanded into another integration technique by replacing those lines. It provides more accurate results by dividing the integration interval [a,b] into n subintervals, and again applying integration rule until achiving desired tolerance in each subinterval. If not met, the subinterval is divided again and the process is done again.

Inputs: lowerlimit - float or integer - integration interval element. upperlimit - float or integer - integration interval element. tol_val - float - desired error tolerance in calculated value. n - integer - numbers of levels or subintervals to generate. function - function type object - evaluates f(x) at x.

Output: app - float - defined integral aproximation.

Example line: adapquad(-2, 4, 0.005, 10, (lambda x: 4*sin(x)**2));

Dependencies: None.

Version: 1.1 for Python 3.4

Definition and structure were taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical differentiation and integration". Cengage Learning. 2010. pp: 220 - 226.

Date: 29/12/2014.

7.2 beziercurve.py

Description: function beziercurve generates an interpolation parametric curve using a third order Hermite polynomial and a pair of nodes and guidelines. Endpoint slope is calculated with the slope of lines tangent to them. Therefore no second or first analytic derivative forms of curve is needed a priori. The fast computation of this interpolant allows an interactive way to generate curves. Spatial coordinates are the only requiremnt for calculation.

Inputs: endpoints - list object - includes 4 float values for nodes. leftguide - list object - includes 4 float values for left node. rightguide - list object - includes 4 float values for right node. n - integer - optional parameter defines number n+1 elements of list.

Output: beziercoeff - list object - Bezier cofficients of polynomial.

Example code: endpoints = [[3.2, 2.4], [3.6, 0.4]]; leftguide = [[2.8, 7.5], [5.3, 6.4]]; rightguide = [[8.3, 7.5], [1.6, 2.4]]; beziercurve (endpoints, leftguide, rightguide, n=10);

Dependencies: None.

Version: 1.3 for Python 3.4

Definition and structure were taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 3 - Interpolation and Polynomial Approximation". Cengage Learning. 2010. pp: 166 - 169.

Date: 27/12/2014.

7.3 bisection.py

bisection.py is a function that calculates the roots of a polinomial function using the bisection numerical method.

inputs= rf ; raw function, string of the form a*x+b a ; left range interval b ; right range interval n ; number of iterations t ; tolerance of calculation

output= x ; the aproximated value of x such that f(x)=0

version: 1.01 for python 2.7

fast_Code: bisection(('6*x-6'),-5,6,200,0.005);

author: JJ.Cadavid - SFTC - EAFIT University date: 24/08/2014.

7.4 clampspline.py

Description: function clampspline generates an interpolant of third order polynomial. Uses knot information of first derivatives in the first and last knot. The coefficients are solved using the tridiagonal linear system vectorial equation AX = B by recursion in the strictly diagonally dominant matrix A.

Inputs: domainpnt - list object - list whose elements are x domain data. imagepnt - list object - list whose elements are y domain data. fstderpnt - list object - first derivative knot information, 2 elements.

Outputs: a, b, c, d - Automatic tuple of lists - interpolant coefficients.

Example code: domainpnt = [0 + (x*(10-0)/5)*x2 for x in range(6)];** imagepnt = [0 + (x*(10-0)/5)*x**(2*3**0.5) for x in range(6)]; fstderpnt = [];

Dependencies: None.

Version: 1.1 for Python 3.4

Definition and structure were taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 3 - Interpolation and Polynomial Approximation". Cengage Learning. 2010. pp: 153 - 156.

Date: 27/12/2014.

7.5 compmidtrule.py

Description: function compmidtrule is part of the Composite Integration Techniques and can be thought as an upgrated version of midpoint rule integration technique that aproximates a defined integal evaluated in [a,b] with better presicion in larger intervals. It's truncation error is of order h^2 and defined by a second order derivative.

Inputs: lowerlimit - float - defines first limit of integration. upperlimit - float - defines last limit of integration. redc - integer - even integer that defines subintervals. function - function type object - evaluates f(x) at x.

Outputs: integ - float - defined integral aproximation

Example line: compmidtrule(-3.15, 6.2, 500, (lambda x: 2*x**3 + 4.3));

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions were taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 203 - 209.

Date: 28/12/2014.

7.6 compsimprule.py

Description: function compsimprule is part of the Composite Integration Techniques and can be thought as an upgrated version of Simpson rule integration technique that aproximates a defined integal evaluated in [a,b] with better presicion in larger intervals. It's truncation error is of order h^4 and defined by a fourth order derivative. Presicion is slighly better than Composite midpoint rule and so is the best option when minimizing number of computations. Is also known to be an all-purpose quadrature algorithm.

Inputs: lowerlimit - float - defines first limit of integration. upperlimit - float - defines last limit of integration. redc - integer - even integer that defines subintervals function - function type object - evaluates f(x) at x.

Outputs: integ - float - defined integral aproximation

Example line: integ = compsimprule(0, 2, 1000, (lambda x: 4*x**3));

Dependencies: None.

Version: 1.2 for Python 3.4

Definition and structure were taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 203 - 209.

Date: 28/12/2014.

7.7 comptraprule.py

Description: function comptraprule is part of the Composite Integration Techniques and can be thought as an upgrated version of Trapezoidal rule integration technique that aproximates a defined integal evaluated in [a,b] with better presicion in larger intervals. It's truncation error is of order h^2 and defined by a second order derivative. Similar to compmidtrule but request more operations Presicion is similar to Composite midpoint rule. Difference from the other methods is the need of only one integration interval therefore the number of subintervals can be even or odd.

Inputs: lowerlimit - float - defines first limit of integration. upperlimit - float - defines last limit of integration. redc - integer - integer that reduces space grid in domain. function - function type object - evaluates f(x) at x.

Outputs: integ - float - defined integral aproximation

Example line: integ = comptraprule(-2, 2, 40, (lambda x: 3*x**4));

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions were taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 153 - 156.

Date: 28/12/2014.

7.8 derivpol.py

Description: function derivpol calculates the derivatives of interpolated data. A good choice to create interpolant are the cubic splines they are used in this program. Clamped and Natural Splines used information of first and second derivatives in knots, by applying algebraic manipulation expressions to calculate derivates are obtained.

Inputs: domainlist - list object - list whose elements are x domain data. imagelist - list object - list whose elements are y domain data. evalpnt - list object - point to evaluate the derivatives.

Outputs: firstderiv, secndderiv - Automatic tuple of lists of derivatives.

Example code: domainlist = [0 + (x*(10-0)/5)*x2 for x in range(6)];** imagelist = [0 + (x*(10-0)/5)*x**4 for x in range(6)]; evalpnt = 5.4; fd, sd = derivpol(domainlist, imagelist, evalpnt);

Dependencies: cubicSpline.py - Jaan Kiusalaasr - 2013.

Version: 1.1 for Python 3.4

Definition and structure were taken from: Jaan Kiusalaasr. "Numerical Methods in Engineering With Python 3". 3th ed. "Chapter 5 - Numerical Differentiation". Cambridge University Press. 2013. PP. 191 - 195.

Date: 28/12/2014.

7.9 FCDA.py

Description: class centraldiff defines a set of numerical methods based on finite differences of central step. Capable of aproximating derivatives at a given point inside an interval. Central step does not evaluate derivatives at endpoints. Precision is highly dependant on interval size, bigger intervals lead to highter truncation errors. Minor error types are of h^2 order. Analitical errors can be calculated using fourth derivatives. Highter derivative orders can lead to high truncation errors, so results are likely to differ a lot from analytical results, especially with interpolated data.

FCDA or First Centered Difference Aproximation, calculates the first derivative in the interval. Dependencies: testlambda, getarray.

Method inputs: n - integer - Defines number of n+1 elements in list. lowerlimit - float - first term of interval. upperlimit - float - last term of interval. function - function type object - evaluates f(x) at x.

Method output: diff_array - list - list of derivated values in (a,b).

call sequence example: centraldiff.FCDA(4, 0, 2, (lambda x: 4*sin(x)**2));

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Jaan Kiusalaasr. "Numerical Methods in Engineering With Python 3". 3th ed. "Chapter 5 - Numerical Differentiation". Cambridge University Press. 2013. PP. 183 - 185.

Date: 28/12/2014.

7.10 fixedpoint.py

fixedpont.py finds the root of a function using the fixed point method given the initial aproximations p0.

Inputs: g - Fixed point function with the form x=g(x) obtained from f(x)=0 p0 - Initial aproximation tol - Tolerance n - Maximum number of itarations

Outputs: Aproximate solution X or message of failure

Quick Code: fixedpoint('(1.0/2.0)*((10.0-x**3)**(1.0/2.0))',1.5,3,10**-8)

Version: 1.0

Author: Sebastián Castaño y Felipe Lopez - SFTC - EAFIT University

7.11 fournodescotes.py

Description: class numecinteg is a set of numerical techniques applied to Numerical Integration, based on the elemental Closed Newton-Cotes and Open Newton-Cotes Formulas. This set holds the fundamental technique for highter ones like Composite Numerical Integration and Adapatative Quadrature. So for that extent, small integration intervals are required to achive certain presicion. The methods described here works in a similar way by taking an integration interval and reduced it in equally spaced elements or more subintervals and sum each of them. The success of this techniques is based on parameter h, which is the space grid size. Lower spacing will grand better presicion, also truncation and round-off error will increase by minor changes in h, every technique differs on how it's analitical error increases.

The program set allows to input interpolated data or an analytical expression as a lambda object. More restriction is applied to interpolated data, since not all orders can be integrated properly with every technique. For high interval integration, comptechrule.py or adapquad.py set is recommended.

Method Inputs: lowerlimit - float - First element in integration interval upperlimit - float - Last element in integration interval function - lambda object type - evaluates f(x) at x – or

- list - coefficients of interpolated data

flag - integer - optional - defines 0: lambda type 1: list type

Method Output: integ - float - Integration aproximation

Example code: integ = newtoncotesf(0, 6, lambda x: 2*x, flag = 0);

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 193 - 201.

Date: 28/12/2014.

7.12 gaussdblint.py

Description: function gaussdblint obtains a defined double integral aproximation using a nested gaussian quarature. The strategy of calculation is using the Legendre polynomials roots and coeficients. So that high grade polynomials can be evaluated with accuracy. Adventage of this is the no need of equally spaced elements in domain. The function allows that the limits of second integral can be variable as a function of the independent variable or can be defined values. The second strategy of calculation is the continuous calculation of the second variable space grid. Since it can change, new grid size is created for every y value.

Inputs: x_limits - list object - list with x first integral limits y_limits - list object - list with y second integral limits or functions evalpnt - list object - point to evaluate the derivatives. m - integer - number of subintervals in y domain. n - integer - number of subintervals in x domain. function - function object type - a double variable lambda object. roots - nested list - list of n, m order roots of Legendre Polynomials. coefflist - nested list - list of n, m coefficients of L. Polynomials. flag - integer - 0: defined y limits, 1: variable y limits. Default 0.

Outputs: J - float - double integration aproximation.

Example code: x_limits = [0, 5]; y_limits = [0, 5]; m = 3; n = 2; function = lambda x,y: y*x*3 + y**2 - x; roots = []; coefflist = [];

Dependencies: None.

Version: 1.3 for Python 3.4

Definition and structure were taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 240 - 244.

Date: 29/12/2014.

7.13 gausstpllint.py

Description: function gausstpllint obtains a defined triple integral aproximation using a nested gaussian quarature. The strategy of calculation is using the Legendre polynomials roots and coeficients. So that high grade polynomials can be evaluated with accuracy. Adventage of this is the no need of equally spaced elements in domain. The function allows that the limits of second or third integral can be variable as a function of the independent variable or can be defined values. The second strategy of calculation is the continuous calculation of the second variable space grid. Since it can change, new grid size is created for every y value and z value.

Inputs: x_limits - list object - list with x first integral limits y_limits - list object - list with y second integral limits or functions z_limits - list object - list with z second integral limits or functions evalpnt - list object - point to evaluate the derivatives. m - integer - number of subintervals in y domain. n - integer - number of subintervals in x domain. p - integer - number of subintervals in x domain. function - function object type - a double variable lambda object. roots - nested list - list of n, m order roots of Legendre Polynomials. coefflist - nested list - list of n, m coefficients of L. Polynomials. flag - integer - 0: defined y limits, 1: variable y limits. Default 0.

Outputs: J - float - double integration aproximation.

Example code: x_limits = [0, 5]; y_limits = [0, 5]; m = 3; n = 2; function = lambda x,y: y*x*3 + y**2 - x; roots = []; coefflist = [];

Dependencies: None.

Version: 1.3 for Python 3.4

Definition and structure were taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 245 - 246.

Date: 29/12/2014.

7.14 hermitpoly.py

Description: function hermitpoly calculates an interpolant based on Hermit interpolation aided with devided difference. Using the first derivative along the domain.

Inputs: domainlist - list object - list whose elements are x domain data. imagelist - list object - list whose elements are y domain data. derivalist - list object - list with first derivative data.

Outputs: hermitcoeff - list object- Hermit interpolant coefficients.

Example code: domainlist = [0 + (x*(10-0)/5)*x2 for x in range(6)];** imagelist = [0 + (x*(10-0)/5)*x**4 for x in range(6)]; derivalist = [];

Dependencies: None.

Version: 1.3 for Python 3.4

Definition and structure were taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 3 - Interpolation and Polynomial Approximation". Cengage Learning. 2010. pp: 133 - 141.

Date: 27/12/2014.

7.15 midpointrule.py

Description: class numecinteg is a set of numerical techniques applied to Numerical Integration, based on the elemental Closed Newton-Cotes and Open Newton-Cotes Formulas. This set holds the fundamental technique for highter ones like Composite Numerical Integration and Adapatative Quadrature. So for that extent, small integration intervals are required to achive certain presicion. The methods described here works in a similar way by taking an integration interval and reduced it in equally spaced elements or more subintervals and sum each of them. The success of this techniques is based on parameter h, which is the space grid size. Lower spacing will grand better presicion, also truncation and round-off error will increase by minor changes in h, every technique differs on how it's analitical error increases.

The program set allows to input interpolated data or an analytical expression as a lambda object. More restriction is applied to interpolated data, since not all orders can be integrated properly with every technique. For high interval integration, comptechrule.py or adapquad.py set is recommended.

Method Inputs: lowerlimit - float - First element in integration interval upperlimit - float - Last element in integration interval function - lambda object type - evaluates f(x) at x – or

  ■ list - coefficients of interpolated data

  flag - integer - optional - defines 0: lambda type 1: list type

Method Output: integ - float - Integration aproximation

Example code: lowerlimit = 0; upperlimit = 5; function = [(x*(10-0)/5)*x-x for x in range(4)]; flag = 1; integ = numecinteg.midpointrule(lowerlimit, upperlimit, function, flag = 0);

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 193 - 201.

Date: 28/12/2014.

7.16 muller.py

Description: function muller is a multiple polynomial root finding algorithm. Due to method limitations functions with roots that are not simple (Multiplicity > 1), Müller's technique is limitated. For that purpose use high order quadrature or adaptative root finding algorithms. Since polynomial roots are unique, all functions of this type, even those with complex roots, can be easily obtained with Müller's algorithm. This program request three seeds in which the roots are located, from there all roots there are obatained. This program will be traslated to polyroot class to make a selection between Müller's and Horner's technique whether or not there are complex roots. If tolerance is not met the program raises an exception.

Inputs: polynm - lambda object type - polynomial function to be evaluated. aprx_list - list object - list containing three seeds. tol - float - error tolerance in roots. iters - integer - Positive value providing the number of loops.

Outputs: p - float - Polynomial root obtained from seeds.

Example code: [];

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 2 - Solutions of Equations in One Variable". Cengage Learning. 2010. pp: 97 - 98.

Date: 24/12/2014.

7.17 natspline.py

Description: function clampspline generates an interpolant of third order polynomial. Uses knot information of second derivatives in the first and last knot. The coefficients are solved using the tridiagonal linear system vectorial equation $AX = B$ by recursion in the strictly diagonally dominant matrix A.

Inputs: domainpnt - list object - list whose elements are x domain data. imagepnt - list object - list whose elements are y domain data. fstderpnt - list object - second derivative knot information, 2 elements.

Outputs: a, b, c, d - Automatic tuple of lists - interpolant coefficients.

Example code: domainpnt = [(x*(10-0)/5)*x2 for x in range(6)];** imagepnt = [(x*(10-0)/5)*x**(2*3**0.5) for x in range(6)]; fstderpnt = [];

Dependencies: None.

Version: 1.1 for Python 3.4

Definition and structure were taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 3 - Interpolation and Polynomial Approximation". Cengage Learning. 2010. pp: 145 - 149.

Date: 27/12/2014.

7.18 nevilletable.py

Description: function nevilletable computes a data interpolant in a recursive way. While Lagrange interpolation request an a priori knowledge of the n order of interpolant, a way of reducing the computation needed to test which polynomial order fits better is done by computing the Neville's Table. Interpolation nodes are created in a similar way such as Newton's divided difference. As an initial aproximation, nodes equal the f(x) data and are slowly force to fit function behaviour by comparing the difference in first x data element with the following element in a forward way. Interval is set dividing the first data with the following element, until is been completed x data reading. When tolerance is not met, new nodes must be include or if so increase tolerance error.

Inputs: domainpnt - list - Elements that compose data x-domain. imagepnt - list - Elements that compose data y-domain. tol - float - Criteria to check whether or not data fits with tolerance

Output: intrpnt - list - Polynomial coefficients of n-th grade

Example code: domainpnt = [(x*(10-0)/5)+x2 for x in range(6)];** imagepnt = [(x*(10-0)/5)*x-x for x in range(6)]; tol = 0.05; intrpnt = nevilletable(domainpnt, imagepnt, tol);

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 3 - Interpolation and Polynomial Approximation". Cengage Learning. 2010. pp: 117 - 123.

Date: 28/12/2014.

7.19 newtondivdef.py

Description: function newtondivdef computes a data interpolant in a recursive way. In a similar way like Neville Table, Newton Divided difference makes a similar approach to fit data by evaluating the function in a certain shifted subset in domain and compare it in the non shifted place - Forward y element respect actual position y element. In a certain way it might look like this approach is similar to the limit definition of first derivative. Coefficients are store in a similar 1-row way done in nevilletable. Tolerance criteria is placed here as well, increase the number of nodes for better fitting or increace tolerance error.

Inputs: domainpnt - list - Elements that compose data x-domain. imagepnt - list - Elements that compose data y-domain. tol - float - Criteria to check whether or not data fits with tolerance

Output: intrpnt - list - Polynomial coefficients of n-th grade

Example code: domainpnt = [(x*(10-0)/5)+x2 for x in range(6)];** imagepnt = [(x*(10-0)/5)*x-x for x in range(6)]; tol = 0.05; intrpnt = nevilletable(domainpnt, imagepnt, tol);

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 3 - Interpolation and Polynomial Approximation". Cengage Learning. 2010. pp: 124 - 126.

Date: 25/12/2014.

7.20 nfirstderv.py

Description: class ncentraldiff defines a set of numerical methods based on finite differences with no central step - forward or backward. Capable of aproximating derivatives at a given point inside an interval including endpoints. Precision is highly dependant on interval size, bigger intervals lead to highter truncation errors. Minor error types are of h^2 order. First non centered aproximations for derivatives are a fast way to obtain a good aproximation, but second non centerd aproximations provides more accurate results but increase computation demand. Other aproximations use high order Taylor series expansion.

Method inputs: n - integer - Defines number of n+1 elements in list. lowerlimit - float - first term of interval. upperlimit - float - last term of interval. function - function type object - evaluates f(x) at x.

Method output: diff_array - list - list of derivated values in (a,b).

call sequence example: function = lambda x: 0.5*x3+5*x*2+6*x+6.33;** nfirstderv(4, 0, 2, function);

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Jaan Kiusalaasr. "Numerical Methods in Engineering With Python 3". 3th ed. "Chapter 5 - Numerical Differentiation". Cambridge University Press. 2013. PP. 185 - 186.

Date: 28/12/2014.

7.21 nseconddderv.py

Description: class ncentraldiff defines a set of numerical methods based on finite differences with no central step - forward or backward. Capable of aproximating derivatives at a given point inside an interval including endpoints. Precision is highly dependant on interval size, bigger intervals lead to highter truncation errors. Minor error types are of h^2 order. First non centered aproximations for derivatives are a fast way to obtain a good aproximation, but second non centerd aproximations provides more accurate results but increase computation demand. Other aproximations use high order Taylor series expansion.

Method inputs: n - integer - Defines number of n+1 elements in list. lowerlimit - float - first term of interval. upperlimit - float - last term of interval. function - function type object - evaluates f(x) at x.

Method output: diff_array - list - list of derivated values in (a,b).

call sequence example: function = lambda x: 0.5*x3+5*x*2+6*x+6.33;** nseconddderv(4, 0, 2, function);

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Jaan Kiusalaasr. "Numerical Methods in Engineering With Python 3". 3th ed. "Chapter 5 - Numerical Differentiation". Cambridge University Press. 2013. PP. 185 - 186.

Date: 28/12/2014.

7.22 onenodeopn.py

Description: class numecinteg is a set of numerical techniques applied to Numerical Integration, based on the elemental Closed Newton-Cotes and Open Newton-Cotes Formulas. This set holds the fundamental technique for highter ones like Composite Numerical Integration and Adapatative Quadrature. So for that extent, small integration intervals are required to achive certain presicion. The methods described here works in a similar way by taking an integration interval and reduced it in equally spaced elements or more subintervals and sum each of them. The success of this techniques is based on parameter h, which is the space grid size. Lower spacing will grand better presicion, also truncation and round-off error will increase by minor changes in h, every technique differs on how it's analitical error increases.

The program set allows to input interpolated data or an analytical expression as a lambda object. More restriction is applied to interpolated data, since not all orders can be integrated properly with every technique. For high interval integration, comptechrule.py or adapquad.py set is recommended.

Method Inputs: lowerlimit - float - First element in integration interval upperlimit - float - Last element in integration interval function - lambda object type - evaluates f(x) at x – or

■ list - coefficients of interpolated data

flag - integer - optional - defines 0: lambda type 1: list type

Method Output: integ - float - Integration aproximation

Example code: lowerlimit = 0; upperlimit = 5; function = [(x*(10-0)/5)*x-x for x in range(4)]; flag = 1; integ = numecinteg.onenodeopn(lowerlimit, upperlimit, function, flag = 0);

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 193 - 201.

Date: 28/12/2014.

7.23 polyroot.py

Description: class numecinteg is a small class type python program used as a prototipe algorithm. It's actual function is to represent a polynomial from a set of coefficients. The second feature is finding it's roots by applying Horner's algorithm.

Method Inputs: None - Input built-in function takes data from user

Method Outputs: Polinomial visualization py,pz,b0 - automatic tuple - Provides information of roots.

Example code: [];

Dependencies: None.

Version: 0.8 for Python 3.4

Definitions are taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 2 - Solutions of Equations in One Variable". Cengage Learning. 2010. pp: 92 - 96.

Date: 24/12/2014.

7.24 QCDA.py

Description: class centraldiff defines a set of numerical methods based on finite differences of central step. Capable of aproximating derivatives at a given point inside an interval. Central step does not evaluate derivatives at endpoints. Precision is highly dependant on interval size, bigger intervals lead to highter truncation errors. Minor error types are of h^2 order. Analitical errors can be calculated using fourth derivatives. Highter derivative orders can lead to high truncation errors, so results are likely to differ a lot from analytical results, especially with interpolated data.

QCDA or Fourth Centered Difference Aproximation, calculates the fourth derivative in the interval. Dependencies: testlambda, getarray.

Method inputs: n - integer - Defines number of n+1 elements in list. lowerlimit - float - first term of interval. upperlimit - float - last term of interval. function - function type object - evaluates f(x) at x.

Method output: diff_array - list - list of derivated values in (a,b).

call sequence example: centraldiff.FCDA(4, 0, 2, (lambda x: 4*sin(x)**2));

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Jaan Kiusalaasr. "Numerical Methods in Engineering With Python 3". 3th ed. "Chapter 5 - Numerical Differentiation". Cambridge University Press. 2013. PP. 183 - 185.

Date: 28/12/2014.

7.25 regulafalsi.py

regulafalsi.py finds the root of a function continuous in an interval [p0,p1] where f(p0) and f(p1) have opposite signs

Inputs: f- Function in terms of x p0,p1 - Interval tol - Tolerance n - Maximum number of itarations

Outputs: Aproximate solution X or message of failure

Quick Code: fixedpoint('(1.0/2.0)*((10.0-x**3)**(1.0/2.0))',1.5,3,10**-8)

Version: 1.0

Author: Sebastián Castaño y Felipe Lopez - SFTC - EAFIT University

7.26 richarexpol.py

Description: function richarexpol is a small method that applies Richardson Extrapolation. This technique is one of the keystones in Advance Numerical Methods. Extrapolation allows to increase the presicion of a numerical result whitout increasing data nodes, whenever space grid size, h, is involve. For this program, extrapolation receives 2 approximations of a value dependent on h, and by setting the correct order p from analytical truncation error high accurate results can be obtained.

Inputs: P - integer - Non negative integer setting the order of correction term firstaprx - float - first approximation of an h dependent value; secondprx - float - second approximation of an h dependent value;

Method Outputs: G - float - High accurate result

Example code: [];

Dependencies: None.

Version: 1.0 for Python 3.4

Definitions are taken from: Jaan Kiusalaasr. "Numerical Methods in Engineering With Python 3". 3th ed. "Chapter 5 - Numerical Differentiation". Cambridge University Press. 2013. PP. 188 - 189.

Date: 28/12/2014.

7.27 romberginteg.py

Description: function romberginteg computes an approximation of a defined integral in [a,b]. Romberg Integration uses other minor techniques to produce accurate results. First aproximations are obtain from one of the Newton-Cotes Formulas and then results are improve with extrapolation techniques. Then the process is done all over again on the subinterval section. Every loop grid size is half for better presicion improvement.

Inputs: lowerlimit - float - First element in integration interval upperlimit - float - Last element in integration interval subinterv - positive integer - Number of subintervals function - lambda object type - evaluates f(x) at x

Outputs: R - list object - Romberg table - integration aproximation.

Example code: R = romberginteg(-3.15, 6.2, 10, (lambda x: 2*x**3 + 4.3));

Dependencies: None.

Version: 1.5 for Python 3.4

Definition and structure were taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 213 - 218.

Date: 19/02/2015.

7.28 SCDA.py

Description: class centraldiff defines a set of numerical methods based on finite differences of central step. Capable of aproximating derivatives at a given point inside an interval. Central step does not evaluate derivatives at endpoints.

Precision is highly dependant on interval size, bigger intervals lead to highter truncation errors. Minor error types are of h^2 order. Analitical errors can be calculated using fourth derivatives. Highter derivative orders can lead to high truncation errors, so results are likely to differ a lot from analytical results, especially with interpolated data.

SCDA or Second Centered Difference Aproximation, calculates the second derivative in the interval. Dependencies: testlambda, getarray.

Method inputs: n - integer - Defines number of n+1 elements in list. lowerlimit - float - first term of interval. up- perlimit - float - last term of interval. function - function type object - evaluates f(x) at x.

Method output: diff_array - list - list of derivated values in (a,b).

call sequence example: centraldiff.FCDA(4, 0, 2, (lambda x: 4*sin(x)**2));

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Jaan Kiusalaasr. "Numerical Methods in Engineering With Python 3". 3th ed. "Chapter 5 - Numerical Differentiation". Cambridge University Press. 2013. PP. 183 - 185.

Date: 28/12/2014.

7.29 secant.py

secant.py finds the root of a function using the Secant method given the initial aproximations p1 and p0 so that the root is a number between them

Inputs: f - Function in terms of 'x' p0,p1 - Initial aproximations tol - Tolerance n - maximum number of iterations

Outputs: Aproximate solution X or message of failure

Quick Code: secant('math.cos(x)-x',0.5,math.pi/4,5*10**-8,20)

Version: 1.0

Author: Sebastián Castaño - SFTC - EAFIT University

7.30 simpdblintg.py

Description: function simpdblintg obtains a defined double integral aproximation using Simpson Integration Rule by reducing intervals into subintervals. Similar to Double Legendre-Gauss Quadrature, calculations are done from inside to outsie, using outside domain to calculate inside integration domain. The sum process in this method takes the results in three stages, the endpoints of the interval, and subinterval results, this last one is devided on even and odd intervals. The heavy process is done within the subintervals by continuously modifying space grid size and by applying Simpson Rule. There is no need for use roots or coeficients from Legendre Polynomials for approximation, making it a bit faster than the Gauss-Legendre Quadrature, yet presicion might decrease if compare with it.

Inputs: x_limits - list object - list with x first integral limits y_limits - list object - list with y second integral limits or functions evalpnt - list object - point to evaluate the derivatives. m - integer - number of subintervals in y domain. n - integer - number of subintervals in x domain. function - function object type - a double variable lambda object. flag - integer - 0: defined y limits, 1: variable y limits. Default 0.

Outputs: J - float - double integration aproximation.

Example code: x_limits = [0, 5]; y_limits = [0, 5]; m = 3; n = 2; function = lambda x,y: y*x*3 + y**2 - x; roots = []; coefflist = [];

Dependencies: None.

Version: 1.3 for Python 3.4

Definition and structure were taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 240 - 244.

Date: 29/12/2014.

7.31 simprule.py

Description: class numecinteg is a set of numerical techniques applied to Numerical Integration, based on the elemental Closed Newton-Cotes and Open Newton-Cotes Formulas. This set holds the fundamental technique for highter ones like Composite Numerical Integration and Adapatative Quadrature. So for that extent, small integration intervals are required to achive certain presicion. The methods described here works in a similar way by taking an integration interval and reduced it in equally spaced elements or more subintervals and sum each of them. The success of this techniques is based on parameter h, which is the space grid size. Lower spacing will grand better presicion, also truncation and round-off error will increase by minor changes in h, every technique differs on how it's analitical error increases.

The program set allows to input interpolated data or an analytical expression as a lambda object. More restriction is applied to interpolated data, since not all orders can be integrated properly with every technique. For high interval integration, comptechrule.py or adapquad.py set is recommended.

Method Inputs: lowerlimit - float - First element in integration interval upperlimit - float - Last element in integration interval function - lambda object type - evaluates f(x) at x – or

- list - coefficients of interpolated data

flag - integer - optional - defines 0: lambda type 1: list type

Method Output: integ - float - Integration aproximation

Example code: lowerlimit = 0; upperlimit = 5; function = [(x*(10-0)/5)*x-x for x in range(4)]; flag = 1; integ = numecinteg.simprule(lowerlimit, upperlimit, function, flag = 0);

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 193 - 201.

Date: 28/12/2014.

7.32 snfirstderv.py

Description: class ncentraldiff defines a set of numerical methods based on finite differences with no central step - forward or backward. Capable of aproximating derivatives at a given point inside an interval including endpoints. Precision is highly dependant on interval size, bigger intervals lead to highter truncation errors. Minor error types are of h^2 order. First non centered aproximations for derivatives are a fast way to obtain a good aproximation, but second non centerd aproximations provides more accurate results but increase computation demand. Other aproximations use high order Taylor series expansion.

Method inputs: n - integer - Defines number of n+1 elements in list. lowerlimit - float - first term of interval. upperlimit - float - last term of interval. function - function type object - evaluates f(x) at x.

Method output: diff_array - list - list of derivated values in (a,b).

call sequence example: function = lambda x: 0.5*x**3+5*x*2+6*x+6.33; ncentraldiff.snfirstderv(4, 0, 2, function);

call sequence example: ncentraldiff.nfirstderv(4, 0, 2, (lambda x: 4*sin(x)**2));

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Jaan Kiusalaasr. "Numerical Methods in Engineering With Python 3". 3th ed. "Chapter 5 - Numerical Differentiation". Cambridge University Press. 2013. PP. 185 - 186.

Date: 28/12/2014.

7.33 TCDA.py

Description: class centraldiff defines a set of numerical methods based on finite differences of central step. Capable of aproximating derivatives at a given point inside an interval. Central step does not evaluate derivatives at endpoints. Precision is highly dependant on interval size, bigger intervals lead to highter truncation errors. Minor error types are of h^2 order. Analitical errors can be calculated using fourth derivatives. Highter derivative orders can lead to high truncation errors, so results are likely to differ a lot from analytical results, especially with interpolated data.

TCDA or Third Centered Difference Aproximation, calculates the third derivative in the interval. Dependencies: testlambda, getarray.

Method inputs: n - integer - Defines number of n+1 elements in list. lowerlimit - float - first term of interval. upperlimit - float - last term of interval. function - function type object - evaluates f(x) at x.

Method output: diff_array - list - list of derivated values in (a,b).

call sequence example: centraldiff.FCDA(4, 0, 2, (lambda x: 4*sin(x)**2));

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Jaan Kiusalaasr. "Numerical Methods in Engineering With Python 3". 3th ed. "Chapter 5 - Numerical Differentiation". Cambridge University Press. 2013. PP. 183 - 185.

Date: 28/12/2014.

7.34 tesimprule.py

Description: class numecinteg is a set of numerical techniques applied to Numerical Integration, based on the elemental Closed Newton-Cotes and Open Newton-Cotes Formulas. This set holds the fundamental technique for highter ones like Composite Numerical Integration and Adapatative Quadrature. So for that extent, small integration intervals are required to achive certain presicion. The methods described here works in a similar way by taking an integration interval and reduced it in equally spaced elements or more subintervals and sum each of them. The success of this techniques is based on parameter h, which is the space grid size. Lower spacing will grand better presicion, also truncation and round-off error will increase by minor changes in h, every technique differs on how it's analitical error increases.

The program set allows to input interpolated data or an analytical expression as a lambda object. More restriction is applied to interpolated data, since not all orders can be integrated properly with every technique. For high interval integration, comptechrule.py or adapquad.py set is recommended.

Method Inputs: lowerlimit - float - First element in integration interval upperlimit - float - Last element in integration interval function - lambda object type - evaluates f(x) at x – or

> ▪ list - coefficients of interpolated data

flag - integer - optional - defines 0: lambda type 1: list type

Method Output: integ - float - Integration aproximation

Example code: integ = tesimprule(0, 2, lambda x: x, 3, flag = 0);

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 193 - 201.

Date: 28/12/2014.

7.35 thrnodeopn.py

Description: class numecinteg is a set of numerical techniques applied to Numerical Integration, based on the elemental Closed Newton-Cotes and Open Newton-Cotes Formulas. This set holds the fundamental technique for highter ones like Composite Numerical Integration and Adapatative Quadrature. So for that extent, small integration intervals are required to achive certain presicion. The methods described here works in a similar way by taking an integration interval and reduced it in equally spaced elements or more subintervals and sum each of them. The success of this techniques is based on parameter h, which is the space grid size. Lower spacing will grand better presicion, also truncation and round-off error will increase by minor changes in h, every technique differs on how it's analitical error increases.

The program set allows to input interpolated data or an analytical expression as a lambda object. More restriction is applied to interpolated data, since not all orders can be integrated properly with every technique. For high interval integration, comptechrule.py or adapquad.py set is recommended.

Method Inputs: lowerlimit - float - First element in integration interval upperlimit - float - Last element in integration interval function - lambda object type - evaluates f(x) at x – or

> ▪ list - coefficients of interpolated data

flag - integer - optional - defines 0: lambda type 1: list type

Method Output: integ - float - Integration aproximation

Example code: lowerlimit = 0; upperlimit = 5; function = [(x*(10-0)/5)*x-x for x in range(4)]; flag = 1; integ = numecinteg.onenodeopn(lowerlimit, upperlimit, function, flag = 0);

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 193 - 201.

Date: 28/12/2014.

7.36 trapzrule.py

Description: class numecinteg is a set of numerical techniques applied to Numerical Integration, based on the elemental Closed Newton-Cotes and Open Newton-Cotes Formulas. This set holds the fundamental technique for highter ones like Composite Numerical Integration and Adapatative Quadrature. So for that extent, small integration intervals are required to achive certain presicion. The methods described here works in a similar way by taking an integration

interval and reduced it in equally spaced elements or more subintervals and sum each of them. The success of this techniques is based on parameter h, which is the space grid size. Lower spacing will grand better presicion, also truncation and round-off error will increase by minor changes in h, every technique differs on how it's analitical error increases.

The program set allows to input interpolated data or an analytical expression as a lambda object. More restriction is applied to interpolated data, since not all orders can be integrated properly with every technique. For high interval integration, comptechrule.py or adapquad.py set is recommended.

Method Inputs: lowerlimit - float - First element in integration interval upperlimit - float - Last element in integration interval function - lambda object type - evaluates f(x) at x – or

- list - coefficients of interpolated data

flag - integer - optional - defines 0: lambda type 1: list type

Method Output: integ - float - Integration aproximation

Example code: integ = trapzrule(0, 2, lambda x: 1.5*x, flag = 0);

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 193 - 201.

Date: 28/12/2014.

7.37 twonodeopn.py

Description: class numecinteg is a set of numerical techniques applied to Numerical Integration, based on the elemental Closed Newton-Cotes and Open Newton-Cotes Formulas. This set holds the fundamental technique for highter ones like Composite Numerical Integration and Adapatative Quadrature. So for that extent, small integration intervals are required to achive certain presicion. The methods described here works in a similar way by taking an integration interval and reduced it in equally spaced elements or more subintervals and sum each of them. The success of this techniques is based on parameter h, which is the space grid size. Lower spacing will grand better presicion, also truncation and round-off error will increase by minor changes in h, every technique differs on how it's analitical error increases.

The program set allows to input interpolated data or an analytical expression as a lambda object. More restriction is applied to interpolated data, since not all orders can be integrated properly with every technique. For high interval integration, comptechrule.py or adapquad.py set is recommended.

Method Inputs: lowerlimit - float - First element in integration interval upperlimit - float - Last element in integration interval function - lambda object type - evaluates f(x) at x – or

- list - coefficients of interpolated data

flag - integer - optional - defines 0: lambda type 1: list type

Method Output: integ - float - Integration aproximation

Example code: lowerlimit = 0; upperlimit = 5; function = [(x*(10-0)/5)*x-x for x in range(4)]; flag = 1; integ = numecinteg.onenodeopn(lowerlimit, upperlimit, function, flag = 0);

Dependencies: None.

Version: 1.2 for Python 3.4

Definitions are taken from: Richard L. Burden, J. Douglas Faires. "Numerical Analysis" 9th ed. "Chapter 4 - Numerical Differentiation and Integration". Cengage Learning. 2010. pp: 193 - 201.

Date: 28/12/2014.