Python program to solve ordinary differential equations

Praciano-Pereira, T *

Sobral Matematica
23 de julho de 2014
préprints da Sobral Matematica
no. 2013.02
Editor Tarcisio Praciano-Pereira
tarcisio@member.ams.org

Resumo

Neste trabalho estou apresentando três classes escritas em python para lidar com operadores diferenciais e encontrar soluções aproximadas de equações diferenciais ordinárias. O trabalho está em andamento mas já é possível obter alguns resultados. O verdadeiro objetivo é fazer simulações com operadores diferenciais, a solução aproximada das equações é um subproduto.

palavras chave: equações diferenciais ordinárias, operador diferencial, programas em python.

In this paper I am presenting three python classes to deal with differential operators and to find approximate solutions of ordinary differential equations. This is work in progress but some results have already proven the work useful. The main goal is to make simulations with differential operators, the approximate solution is a byproduct.

keywords: differential operator, ordinary differential equations, python programs

*tarcisio@member.ams.org

1 Basic functions

This paper is a manual to explain the functions of the modules used in the python program ODEWithPython.py and at the same an invitation to collaborative work or a more ambitious goal, to expose the work to *nice critic* people. If you come up to this paper to understand the the classes used in my programs, jump to the section 3 where they are directly explained.

The classes directly involved with the project are

`ambiente, funcoes.py, gnuplot.py, nucleo, operadores.py,`

which can be download from [2]. These programs are distributed under `GPL` with the version of your choice.

Let me introduce a short cut in the notation, as we are using to often the function $\chi_{[0,1]}$ we will denote it with only χ.

Some of the methods in these modules have been constructed in the paper [1] in which we have described an algorithmic construction of the powers of χ and we are using here one of these convolution power of χ, which is a kernel with support $[0, n]$, named `Qui()` and defined at `funcoes.py`. As we need an *atom* which is a parcel of the partition of the unity to construct an interpolation projector, the following mathematical operations are needed and are translated into algorithms at `funcoes.py`:

$$\eta(x) = 5Qui(5x);\ \text{to have support } [0,1] \tag{1}$$

$$pre\_atom(x) = \eta * \chi;\ \text{to have support } [0,2] \tag{2}$$

$$atom(x) = pre\_atom(x+1) = atom_0;\ \text{to have support } [-1,1] \tag{3}$$

Of course, we could have defined $atom(x) = 5 * Qui(5x) * \chi$ to have an atom with support $[0, 2]$ but it is easier to break down this operation into small steps to have all calculations done correctly, and more over, we can test the intermediate results looking to the graphic.

We have tried to use a computer algebra program but the results were not satisfactory to our particular case, so the work has been done with *brute force Calculus level integrations*, instead. We have to acknowledge that we have been using computer help heavily since the integrations were mainly done by using the facilities of text edition and to verify that the integration was correctly done we have used `gnuplot` to make the graphic in each and every single step, this explain the name of one of our modules `gnuplot.py` used to call `gnuplot` from inside our programs.

In fact you can see all the work done at the python modules mentioned above as we have not cleaned up the intermediary results and leaving many of them as comments.

Again, to have the convolution $\eta * \chi$ it is easier to calculate its derivative and then the primitive:

$$(\eta * \chi)' = \eta * (\delta_0 - \delta_1) = \eta_0 - \eta_1 = \eta - \eta_1; \tag{4}$$

$$atom(x) = \int_0^x (\eta * \chi)'(t)dt = (\eta * \chi)(x) \tag{5}$$

this is an easy trick that we have learned when working at [1].

The integration of the last equation is easier to calculate than the one of the convolution in the first equation. This explains the work done at `funcoes.py` to produce `atom()`. Remark that η_0 and η_1 have disjoint support hence the integral of their difference is the difference between their integrals. This happens to be consequence of the selection of the support of η.

The work is done for an integer partition of the interval $[0, n]$ which can be later translated to any uniform partition of an arbitrary interval $[a, b]$ using wavelet-type dilation and translations.

Since η is symmetric around the central point of its support 0.5 then $atom = \eta * \chi$ is symmetric around the central point of its support 1, $support(atom) = [0, 2]$, and the summation over a set of n translations $(atom_k)_{k=-1}^{n-1}$ may be written putting η in factor, as convolution factor, of the summation of the translations $(\chi_k)_{k=-1}^{n-1}$. This summation of integers translations of χ is an almost constant function, almost always 1, with discontinuities of first kind on the integers of the interval $[0, n]$ where it jumps to 2 hence the convolution with η regularize it to the same class of continuity of η which is a 5-splines. This proves

Theorem **1 (Partition of the unity)** *Partition of the unity The integer translations $(atom_k)_{k=-1}^{n-1}$ form a partition of the unity of class $\mathcal{C}^4$ whose elements are 5-splines subordinated to to the open cover of $((k-1, k+1))_{k=0}^{n}$ of $[0, n]$.*

The functions members of the partition of the unity described by theorem 1 being symmetric around the central point of its support have the value 1 on this central point. By construction their supports expand over two successive intervals, this makes the summation

$$\phi(f)(x) = \sum_{k=0}^{n} f(k) atom_k(x) \tag{6}$$

resumed to exactly two successive *atoms* in whose support x lies. So, no matter how big may n be we have a fast algorithm.

2 Proof of convergence

We though we have found a more advanced method, to calculate approximate integrals. This is indeed false as the following proof will show, but as a collateral and nice result, we can prove the convergence of the algorithm.

To express formally the integral of the projection is easy and straightforward, as $\Phi(f)$ is a linear combination of translates of the 5-splines

$$\texttt{atom} = (\texttt{x} \mapsto \texttt{5Qui(x)}) * \chi \tag{7}$$

where we are using lambda-function-style-syntax at equation (eq. 7) to "remove" the variable x of the expression $5Qui(x)$). Hence there is only one integral to calculate as far as we are dealing with uniform partitions (and in case

we were not, there would be a wavelet-type-change-of-variable, again reducing the integrals to only one).

$$x \in [x_{N+1}, x_{N+2}];\ \int_a^x PU(f)(t)dt = \tag{8}$$

$$= \int_a^x \sum_{k=0}^{N} f(x_{k+1}) \int_{x_k}^{x_{k+1}} atom_w(t)dt + f(x_{N+2}) \int_{x_{N+1}}^{x} atom_w(t)dt = \tag{9}$$

$$= \sum_{k=0}^{N} f(x_{k+1})A_w + f(x_{N+2}) \int_{x_{N+1}}^{x} atom_w(t)dt = \tag{10}$$

$$= A_w \sum_{k=0}^{N} f(x_{k+1}) + f(x_{N+2}) \int_{x_{N+1}}^{x} atom_w(t)dt \tag{11}$$

proving that this is equivalent to Riemann sums, where A_w is the integral of *atom* under dilation to a subinterval of the partition of $[a, b]$ under consideration.

As this is equivalent to Riemann sums then its use will not provide a better algorithm to calculate approximate integral, at least not better than the know algorithms. So we have switched to the previous operator `RiemSpl()` in this work.

Incidentally this proves a kind of convergence of the algorithm we are constructing. It remains to associate this calculus to Sobolev type of convergence which will be our future work which is supported by the computational results we have reached and displayed at the end of the paper.

3 The modules

Next we shall explain the methods in each of the modules of the program. The main program is `executa.py` which is going to use the modules `gnuplot.py, operadores.py, nucleo.py`. We shall not discuss `ambiente.py, funcoes.py` because we think they are self explanatory and, at the end, you can use the methods defined in them without understanding them as if they were *python commands*, anyhow we are convinced there is enough explanations through the comments, be sure, you are welcome with your questions about these classes and we are quite sure of our gains with your point of view.

3.1 `gnuplot.py`

This a service module which is used to help with graphics any program which may need. The user need to know of

grafun(),grafun2(),grafun3(),grafun4(),grafun5()

where `grafun()` may plot the graphic of one function and `grafun5()` may do it with 5 functions. We have to call the appropriate method and the syntax is, in case of `grafun3()`,

```
grafun3(f1,f2,f3, init, end, number_points, message);
```

to plot the graphics of three functions on the interval $[init, end]$ with the positive integer `number_points`, using `message` as title, which has to be a string. Similarly for the other `grafun-methods`. It is going to call two other internal methods to produce the temporary files `dados1, dados2, dados3` each with the matrix to plot the graphic of the corresponding function and the file `transfere3` with the appropriate commands for `gnuplot`. This way you can replot the graphic issuing the command:

```
gnuplot transfere3
```

at a terminal window. You can edit these files and rename them for later as each time you will run the program they be rewritten. All you have to do, to use the methods of `gnuplot.py`, is to include the commands

```
from gnuplot import *
```

at beginning of your python program. See `executa.py` as an example. But be aware that this program is in fact a `script` so its version at [2] can be changed at any moment. The objective with it is exactly to call in ordered way the methods to produce an algorithm.

3.2 nucleo.py

Contains the definitions of kernels, basically two, `Qui()` which is the fifth convolution power of $\chi_{[0,1]}$ and its four successive derivatives

```
d1_Qui(), d2_Qui(),d3_Qui(),d4_Qui()
```

`eta, rho, atom` as described in the initial section.

There are several methods which are need do define `Qui()` which we have borrowed from the program [2, `convolution_power.calc`] needed to construct the convolution powers of $\chi_{[0,1]}$ but to reproduce the work in this paper all you need is to run `main(n)` with the appropriate value for n and `convolution_power.calc` will print the matrix of this power to be used in a python program in a manner similar we are doing here with `Qui()`. Put this power instead of `Qui()` and you are done. Questions are welcome.

3.3 operadores.py

This is perhaps the best part of this work putting aside the construction of convolution powers which is the underlining work. Most of the operators defined in this module are self explained with comments, let us explain two of them which are directly used in this work.

3.3.1 RiemSpl

This a method to calculate approximate integrals using quasi-splines, which are piecewise degree three polynomials whose derivative is not necessarily continuous. It is well suited to calculate approximate integrals but we are going to use it directly here. It uses a number, internally, to partition the interval and creates the degree three polynomial with help of the convolution derivative at the node points. We have used this in previous version of this work and it was

proven to be very fast to calculate primitives. So to heat up your appetite to understand `RiemSpl` let me put here the basic use of this method in our work:

$$\texttt{F(x) := RiemSpl(f, a, x);} \tag{12}$$

$$F(x) = \int_a^x f(t)dt; \tag{13}$$

where (eq. 3.12) is a *fake* `python` command to define $F(x)$ but which shows the Mathematics which is genuinely put at equation (eq. 3.13), and you may want to try

`grafun2(F,f, a,b, 1000, "f and one of its primitives F");`

or perhaps best

`grafun3(G, F,f, a,b, 1000, "f and one of its primitives F");`

to have the program to plot G, F, f where G is a primitive of f which you have taken from an integral table. The number $\widehat{1000}$ will be number of points that `grafun3()` will use to make a uniform partition of $[a, b]$ to create the matrix for the graphic and `RiemSpl()` will use an internal constant, usually `10`, to split further each subinterval to calculate three degree polynomial approximations of F. We bet will be surprised.

4 Computer results

At the time of this writing,10:58 of Monday, October 21 2013, we have ran the program at home computer starting at 08:44 of Monday, October 21 2013, the program has been running for roughly 1:47 hours (47 minutes), this proves that either we are working at a very weak computer, which is indeed true, or that the program has to be optimized and we are also sure that this is true.

The results obtained were:

1. The graphic at the figure figura (Fig. 1) page 6, where the graphics of $f, Riem_d f, PU_d f, df$ are displayed and let me rapidly describe the meaning of this notation,

   (a) $f(x) = e^x$ is the data in the differential equation solved by the program as proposed at the beginning of the paper.

   (b) $Riem_d f$ is the output of the operator `Riem()` which was our option in this paper to calculate approximate integrals. This is primitive of the solution of the equation obtained with Riemann sums. This is one of the reasons of the law performance of the program. The module calculating primitive using splines is not working, we found some bugs in it recently.

   (c) $PU_d f$ is the output of the operator "partition of the unity projector" applied to the solution of the equation, hence this produces an approximation of the solution.

   (d) df is the formal (exact) derivative of f.

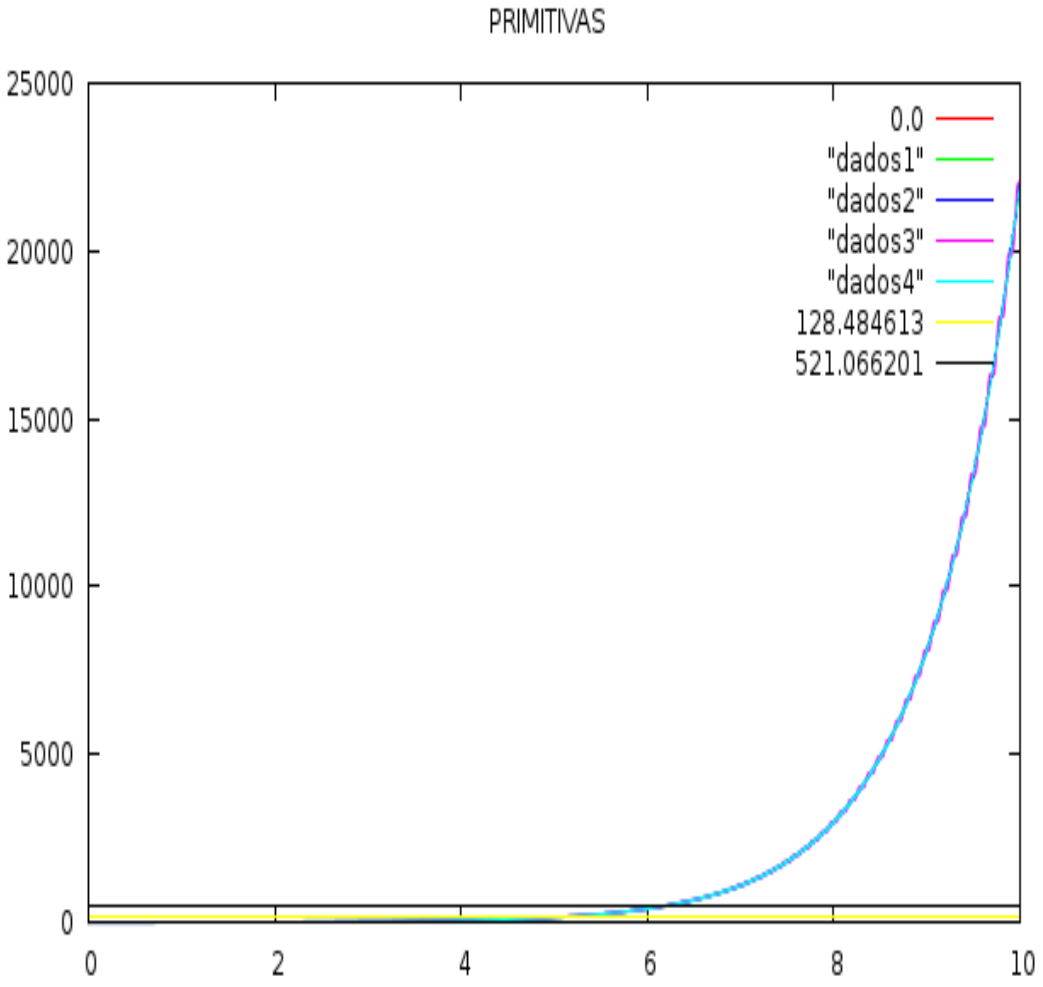

Figura 1: Four graphics

2. the figure (Fig. 2) page 7, displays a very near zoom of the previous image showing how the work has been done. The error is very big as expected by the choice of the equation, remember that this work is to test the method.

3. Calculus of the norm:

   `Norma = || 127.386952 , 521.141738 || = 648.528690`

   this is the output of the program. In each cycle, in the construction of the graphics, the maximum value of the difference of respective pairs, $(f, Riem_d f)$ and $(PU_d f, df)$ among the four functions has been calculated and finally written down at the file containing the commands for `gnuplot` making me possible to retrieve the value to put here.

   The python programs cited here are published at [2, ODEWithPython.tgz] you can find them with the names listed here. The version mentioned here

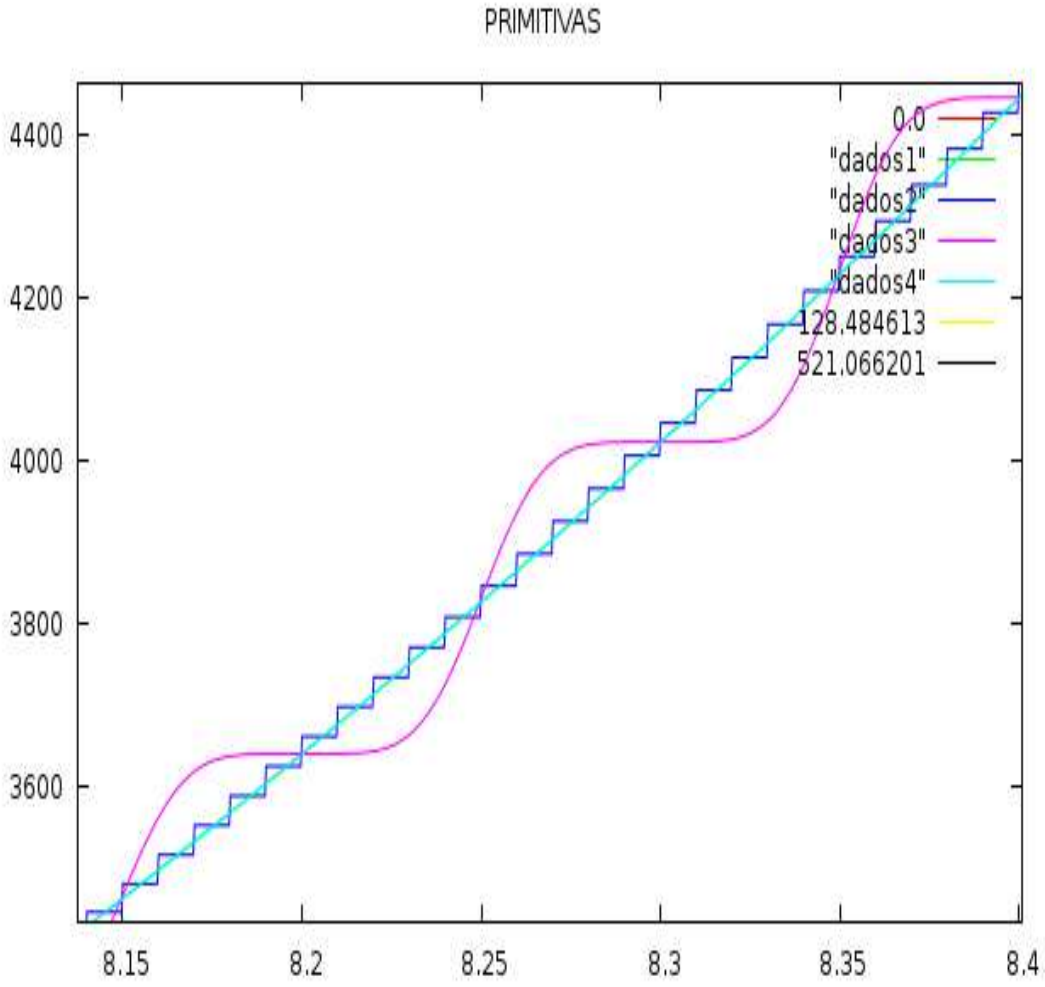

Figura 2: Zoomed image of "four graphics"

is not last one, this is work in progress.

Referências

[1] A.J. Neves and T. Praciano-Pereira. Convolutions power of a characteristic function. *arxiv.org*, 2012, April, 22:16, 2012.

[2] Tarcisio Praciano-Pereira. Programas para cálculo numérico. Technical report, 2009. http://www.calculo-numerico.sobralmatematica.org/programas/.