Cantor: Ambiente Integrado de Desenvolvimento Voltado à Computação Científica Utilizando Python

Filipe de Oliveira Saraiva[1]

[1]Escola de Engenharia de São Carlos – Universidade de São Paulo (USP)
Av. Trabalhador São-carlense n. 400 – 13566-590
São Carlos – SP – Brasil

filipe.saraiva@usp.br

Abstract. *Scientific computing is the study area focused on development and evaluation of models and numerical techniques as software applied on solving and analyzing mathematical, scientific, and engineering problems. The use of scripting languages is common in this area because it provide easy development and validation in prototyping. As important as a good programming language, a good integrated development environment can increase productivity and facilitate tasks completion for this type of application. In this paper will be presented the Cantor software, an integrated development environment to scientific computing. The paper will focus in the Cantor support to Python programming language – this feature was developed by the author of this paper.*

Resumo. *Computação científica é a área de estudos focada na construção e avaliação de modelos e técnicas numéricas em forma de software, voltadas para a solução e análise de problemas matemáticos, científicos e de engenharia. É comum a utilização de linguagens de script para esse propósito, pois estas proporcionam rápido desenvolvimento e validação na criação de protótipos. Tanto quanto a linguagem de programação, um bom ambiente integrado de desenvolvimento pode incrementar a produtividade e facilitar a realização de tarefas para esse tipo de aplicação. No presente artigo será apresentado o software Cantor, um ambiente integrado de desenvolvimento voltado para a computação científica. O trabalho se focará no suporte do Cantor à linguagem de programação Python – funcionalidade desenvolvida pelo autor do artigo.*

1. Introdução

Matemáticos, físicos, químicos, cientistas da computação, engenheiros, e outros pesquisadores de determinados ramos científicos com muita afinidade com processamento de cálculos, se utilizam do desenvolvimento de software para modelar e implementar métodos numéricos para solução e análise de diferentes problemas encontrados em seus respectivos campos de estudo. Convencionou-se chamar esse tipo de utilização e desenvolvimento de software de computação científica [Michael 2002], também sendo comum a utilização dos termos programação científica, programação matemática, e outras.

Se por um lado é comum a utilização de linguagens compiladas nesses tipos de aplicações, conhecidas pela alta performance e velocidade de cálculo, como C e Fortran, empregadas principalmente para problemas com alto custo computacional e que exigem um grande número de cálculos e iterações, por outro lado é também comum a utilização

de linguagens do tipo *script*, de mais alto nível, que não tem um desempenho tão bom quanto os das linguagens compiladas citadas anteriormente, porém permitem uma rápida prototipação do software acelerando a possibilidade de realização de verificações e testes, além de facilitar a correção de erros tanto no sistema desenvolvido quanto no modelo.

Para esta segunda característica, há vários tipos de linguagens e plataformas disponíveis para este tipo de trabalho. Certamente, a mais conhecida e utilizada é a plataforma proprietária Matlab [Moler 2008, Inc. 2010], da MathWorks, que entrega tanto uma linguagem de programação quanto um ambiente integrado de desenvolvimento, além de diversos plugins com funcionalidades específicas chamados na ferramenta de *toolboxes*.

Entretanto, há também diversas alternativas em software livre para esse tipo de aplicação que entregam funcionalidades similares às encontradas na popular ferramenta proprietária citada. Entre estas alternativas, é possível enumerar o Scilab [Scilab Enterprises 2012, Gomez 1999], GNU Octave, Maxima, R, Sage, e outras.

Além destas, a linguagem de programação Python[1] vem se destacando nesse tipo de atividade, principalmente por conta de sua comunidade, bastante ativa e atuante, que desenvolve diversos pacotes para diferentes funcionalidades, além de organizarem fóruns específicos sobre a utilização da linguagem na computação científica.

Porém, apesar do Python prover tanto a linguagem de programação quanto os pacotes que facilitam sua utilização, ela não oferece por padrão um ambiente integrado de desenvolvimento que facilite as tarefas de programação de software típicos da computação científica.

Para este fim, foi desenvolvido o software Cantor[2], software livre disponível sob a licença GPLv2, que entrega um ambiente de desenvolvimento voltado para a programação científica. A disponibilidade de ferramentas que facilitam o processo de desenvolvimento, em especial na área de computação científica, podem fomentar sobremaneira a aceitação e utilização de uma plataforma [Prabhu et al. 2011].

Este artigo descreverá a utilização da linguagem de programação Python em atividades de computação científica, utilizando o Cantor como ambiente de programação. O suporte ao Python no Cantor foi desenvolvido pelo autor do presente artigo. O texto focará nas funcionalidades e facilidades fornecidas por esse software para esse tipo de aplicação.

O restante do texto está assim dividido: Seção 2 descreverá a utilização de Python como linguagem de programação para computação científica, destacando os pacotes mais utilizados nesse tipo de programação; Seção 3 descreverá o software Cantor de forma geral, sua arquitetura e a tecnologia utilizada no seu desenvolvimento. Seção 4 apresentará o desenvolvimento do *backend* que possibilita a comunicação entre o Cantor e Python, permitindo a utilização da linguagem nesse ambiente de desenvolvimento. Seção 5 descreverá as funcionalidades existentes no Cantor para utilização do Python, e a Seção 6 mostrará um exemplo de uso a partir do desenvolvimento de redes neurais artificiais. Finalmente, a Seção 7 apresentará as conclusões e trabalhos futuros do estudo desenvolvido.

[1]Informações sobre a linguagem Python podem ser obtidas em `http://python.org` (Acessado dia 24 de março de 2014).

[2]Maiores informações sobre o Cantor podem ser obtidas em `http://edu.kde.org/cantor` (Acessado dia 24 de março de 2014).

2. Computação Científica com Python

Python é uma linguagem de programação de alto nível, interpretada, de propósito geral. Há aplicações de diversos tipos que usam Python, tanto em ambiente desktop quanto na web, e há *frameworks* e pacotes para diversas finalidades.

Python passou a ser bastante utilizada em computação científica, principalmente por conta da disponibilidade de uma ampla gama de pacotes para variados tipos de aplicações nessa área.

Os pacotes que facilitam a utilização de Python em computação científica estão reunidos sob a assim chamada pilha SciPy (sigla para a expressão em língua inglesa *Scientific Python*). Cabe ressaltar que a palavra SciPy identifica diferentes entidades na comunidade científica de utilizadores Python: ela serve para descrever o conjunto de bibliotecas científicas; nomeia uma biblioteca específica recheada de funções para programação numérica; nomeia as conferências dedicadas à utilização de Python em programação científica; e também identifica a própria comunidade científica de utilizadores de Python[3].

Os principais pacotes que compõe a pilha SciPy [Oliphant 2007, Hunter 2007, Pérez and Granger 2007] são:

- Python – a própria linguagem de programação Python;
- SciPy – biblioteca de funções numéricas;
- NumPy – biblioteca com a definição de tipos de array e matrizes, bem como suas respectivas operações;
- Matplotlib – biblioteca popular para produção de gráficos de alta qualidade;
- pandas – conjunto de estruturas de dados de fácil utilização;
- SymPy – matemática simbólica e álgebra computacional;
- IPython – interface interativa com funções adicionais para computação científica.

Além dos pacotes listados há muitos outros disponíveis, como o Mayavi para criação de gráficos tridimensionais, o Chaco também para criação de gráficos, além de outros mais específicos como o scikit-learn para aprendizado de máquina, spade2 para desenvolvimento de sistemas multiagentes, e muito mais.

A Figura 1 apresenta um exemplo simples de computação com matrizes utilizando a biblioteca NumPy. No caso, foram definidas duas matrizes, “a” e “b”, e em seguida foram realizadas operações de adição, multiplicação, e multiplicação das transpostas das matrizes criadas.

Já a Figura 2 apresenta uma imagem da função de dois gráficos, a saber, a função seno (representada pela cor azul) e a cosseno (representada pela cor verde) de “x”, variando no intervalo de 0 à 10. Esse gráfico foi produzido utilizando a biblioteca Matplotlib.

São essas e outras bibliotecas que, aliadas à linguagem Python, permitem a realização de aplicações com sucesso dessa linguagem de programação na computação científica.

A próxima seção descreverá o software Cantor, em especial suas características e arquitetura.

[3]O site específico da comunidade de utilizadores de Python em computação científica é `http://scipy.org` (Acessado dia 24 de março de 2014)

```
$ python
Python 2.7.6 (default, Feb 15 2014, 14:56:26)
[GCC 4.8.2] on linux2
>>> import numpy
>>> a = numpy.matrix([[1, 2],[3, 4]])
>>> b = numpy.matrix([[5, 6],[7, 8]])
>>> a + b
matrix([[ 6,  8],
        [10, 12]])
>>> a * b
matrix([[19, 22],
        [43, 50]])
>>> a.transpose() * b.transpose()
matrix([[23, 31],
        [34, 46]])
```

Figura 1. Operações de matrizes utilizando NumPy

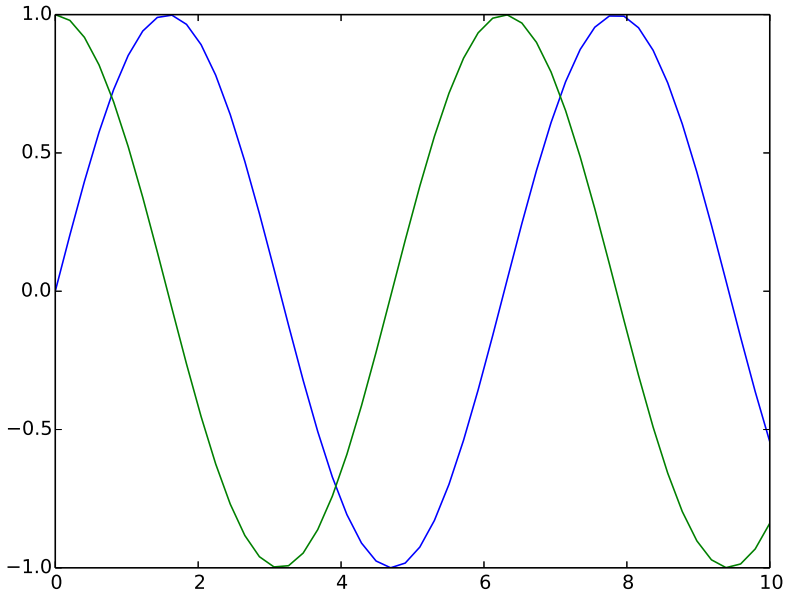

Figura 2. Gráfico produzido com Matplotlib

3. Cantor – Ambiente Integrado de Desenvolvimento para Computação Científica

Cantor é um software livre, disponibilizado sob licença GPLv2, desenvolvido pela comunidade KDE[4] e parte integrante do projeto de softwares educacionais KDE-Edu[5]. Seu nome faz alusão ao matemático Georg Cantor, conhecido principalmente pelo desenvolvimento da teoria dos conjuntos. O software foi criado pelo desenvolvedor Alexander Rieder, em 2009, e desde então tem tido suas funcionalidades expandidas por diversos outros programadores.

Cantor é praticamente todo escrito em C++, utilizando as bibliotecas Qt[6] e o *framework* de desenvolvimento provido pela comunidade KDE, a KDE-libs. Entretanto, há a utilização de diversas APIs e bibliotecas adicionais que permitem ao software utilizar diferentes linguagens de programação.

O software Cantor pode ser definido como um ambiente integrado de desenvolvimento (do inglês *Integrated Development Environment* – IDE) que entrega diversas facilidades para programação em linguagens que tem como alvo a utilização em computação científica.

A arquitetura do software segue um modelo de divisão dos componentes em uma série de *plugins*, tornando fácil o engajamento do desenvolvedor em uma determinada funcionalidade, sem a necessidade de conhecer o código de outras partes do programa para realizar contribuições.

Essa característica é valida tanto para a criação de infraestruturas – ou *backends* – que permitem ao Cantor suportar diferentes linguagens de programação, quanto para as funcionalidades fornecidas enquanto ambiente de desenvolvimento.

Atualmente, o Cantor suporta diferentes linguagens de *script*: GNU Octave, R, Sage, Maxima, KAlgebra, Qalculate, Scilab e Python. Os suportes para as duas últimas linguagens de programação da lista, Scilab e Python, foram desenvolvidos pelo autor do artigo.

A Figura 3 apresenta o menu de seleção das linguagens suportadas, a tela inicial apresentada logo após a execução do software Cantor.

Para escrita do suporte à novas linguagens, o Cantor exige um código que possibilite a comunicação entre um programa em C++ (o próprio Cantor) e a linguagem alvo. Esse código normalmente remete ao uso de alguma API da linguagem alvo, mas também é possível implementar tal comunicação utilizando troca de mensagens entre processos. Independente da forma escolhida, o resultado é que entradas realizadas no Cantor são enviadas para a linguagem alvo, que processam a entrada e emitem a resposta, capturada em seguida pelo Cantor e apresentada ao usuário.

Com essa comunicação implementada, é possível desenvolver as diferentes funcionalidades possíveis que formam o ambiente integrado de desenvolvimento. Todas elas

[4]Informações sobre a comunidade KDE em `http://kde.org` (Acessado dia 24 de março de 2014)

[5]Informações sobre o projeto KDE-Edu em `http://edu.kde.org` (Acessado dia 24 de março de 2014)

[6]Informações sobre a biblioteca Qt em `http://qt-project.org` (Acessado dia 24 de março de 2014)

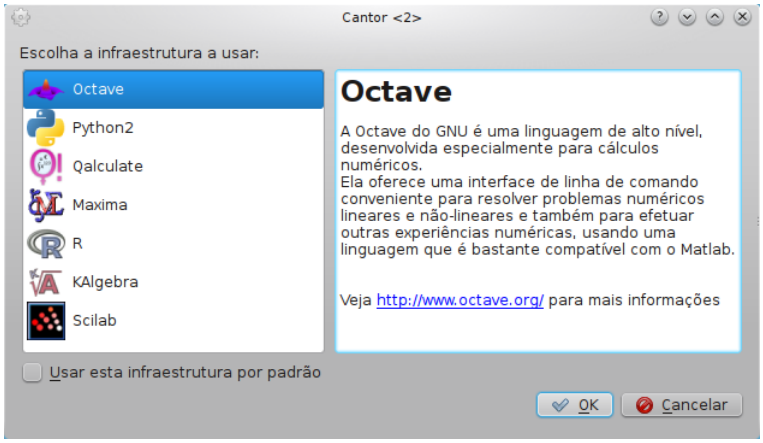

Figura 3. Tela de seleção das linguagens suportadas

são disponibilizadas a partir de *plugins*, e caberá ao desenvolvedor escrever o código específico que habilitará a funcionalidade para a linguagem desejada.

Entre essas funcionalidades, destacam-se:

- terminal integrado ao ambiente;
- destaque de sintaxe;
- complementação de código;
- painel para gerenciamento de variáveis;
- painel para exibição de ajuda;
- editor de *script*;
- inclusão de gráficos no próprio terminal;
- acesso facilitado à funções de álgebra linear, gráficos, etc;

As funcionalidades presentes no suporte à linguagem Python serão apresentadas na Seção 5.

Com as funcionalidades e arquitetura do software Cantor descritas, a próxima parte desse artigo, Seção 4, apresentará como a comunicação entre o software e o Python foi desenvolvida.

4. Comunicação entre Cantor e Python

Para o desenvolvimento da comunicação entre Cantor e Python foi utilizada a API Python/C[7], presente por padrão na linguagem de *script*.

Essa API permite, a partir de um código C ou C++, instanciar um processo do interpretador Python e realizar a comunicação de duas vias entre os processos – o código C/C++ envia comandos ao interpretador Python, e em seguida o resultado da computação pode ser obtido utilizando comandos providos pela API no próprio código C/C++.

Em termos de código, após adicionar o arquivo de cabeçalho "Python.h" em um código C/C++, inicializa-se o processo do interpretador Python utilizando

[7]Informações sobre a a API Python/C disponíveis em `http://docs.python.org/2/c-api` (Acessado dia 24 de março de 2014)

a diretiva “Py_Initialize()” seguido da expressão “PyObject *m_pModule = PyImport_AddModule(“__main__”)”. Essa última serve para referenciar o módulo Python que se está trabalhando no interpretador, no caso, o “main”.

Em seguida, utiliza-se o comando “PyRun_SimpleString(const char*)” para enviar as entradas a serem processadas pelo Python. O argumento dessa função é um ponteiro para um vetor de caracteres, que deve ser o comando que se quer processar.

Porém, antes de enviar o comando propriamente, decidiu-se fazer um pré-processamento para facilitar a captura das saídas do comando. A Figura 4 apresenta uma classe Python, nomeada “CatchOutPythonBackend”, e uma série de comandos que redirecionam *stream* padrões do sistema operacional para variáveis no Python.

```
import sys
class CatchOutPythonBackend:
    def __init__(self):
        self.value = ''
    def write(self, txt):
        self.value += txt
outputPythonBackend = CatchOutPythonBackend()
errorPythonBackend  = CatchOutPythonBackend()
sys.stdout = outputPythonBackend
sys.stderr = errorPythonBackend
```

Figura 4. Classe que redireciona as saídas e erros dos comandos Python

O que o código apresentado realiza pode ser descrito por: cria-se a classe “CatchOutPythonBackend” que adiciona textos para a variável “value”. Em seguida, são instanciados dois objetos dessa classe – “outputPythonBackend” e “errorPythonBackend”. As duas últimas linhas redirecionam saídas *stream* padrões para estes objetos – no caso, a saída padrão (*stdout*) é direcionada para “outputPythonBackend”, enquanto a saída de erros padrão (*stderr*) é direcionada para o objeto “errorPythonBackend”.

Ou seja, todas as saídas dos comandos enviados, tanto as corretas quanto os erros, serão guardadas de forma separada, em objetos próprios para cada uma.

Essa classe é então enviada para o interpretador Python utilizando o comando já comentado, “PyRun_SimpleString(const char*)”. Após o processamento dela, os comandos que de fato queremos ver sendo processados serão enviados utilizando também esta função.

Após o processamento, a API Python/C fornece comandos para capturar variáveis do ambiente Python. É possível utilizar a função “PyObject_GetAttrString(PyObject*, const char*)” para extrair um atributo, referenciado no segundo argumento da função, de um “PyObject”, referenciado no primeito argumento. O retorno dessa função será também um objeto “PyObject”.

Utiliza-se a função descrita anteriormente para extrair um objeto do módulo que se está trabalhando no Python. A seguir, a mesma função foi utilizada novamente, agora para extrair um atributo do objeto previamente capturado.

Com esse novo atributo recuperado, agora é possível utilizar a função "PyString_AsString(PyObject*)" que transforma o "PyObject" em uma *string*, conforme definida pela biblioteca padrão STD do C++.

A Figura 5 apresenta a utilização das funções previamente descritas para capturar as saídas corretas e erros da computação de comandos enviados para o interpretador Python.

```
PyObject *outputPython = PyObject_GetAttrString(m_pModule,
  "outputPythonBackend");
PyObject *output = PyObject_GetAttrString(outputPython,
  "value");
string outputString = PyString_AsString(output);

PyObject *errorPython = PyObject_GetAttrString(m_pModule,
  "errorPythonBackend");
PyObject *error = PyObject_GetAttrString(errorPython,
  "value");
string errorString = PyString_AsString(error);
```

Figura 5. Captura de saídas corretas e erros do Python em código C/C++

Com os procedimentos descritos, é possível realizar a comunicação entre software que utiliza código C ou C++ e um interpretador Python. Com as saídas das computações obtidas, é possível tratá-las adequadamente no software utilizado para a interação com o usuário.

Esses procedimentos foram utilizados no Cantor, permitindo o desenvolvimento do suporte para Python nesse ambiente de desenvolvimento.

A partir desta etapa, foi possível desenvolver o suporte para as demais funcionalidades do Cantor que adicionam facilidades para a utilização deste software e a linguagem Python no campo da computação científica. A Seção 5 descreverá estas funcionalidades.

5. Funcionalidades do Cantor Presentes no Suporte à Python

Após iniciar o Cantor e selecionar a infraestrutura Python, a tela inicial do software se apresentará como mostrado na Figura 6. É possível perceber uma grande área com um *prompt*, representando um terminal, e alguns painéis laterais cujas abas indicam as funcionalidades: *Ajuda* e *Gerenciador de Variáveis*.

Clicando em *Configurações* $\Rightarrow$ *Configurar o Cantor* $\Rightarrow$ *Python2*, é possível encontrar um painel com algumas opções: indicar o caminho para o executável do Python; selecionar se os gráficos gerados serão apresentados no terminal ou fora dele, além de ser possível também definir uma série de comandos a serem executados na inicialização do Cantor quando selecionada a infraestrutura Python. A Figura 7 apresenta essa tela, incluindo dois pacotes configurados para serem carregados na inicialização do software.

Retornando à tela principal do Cantor, realizou-se uma série de operações cujos resultados podem ser observados na Figura 8. Foram criadas uma série de variáveis,

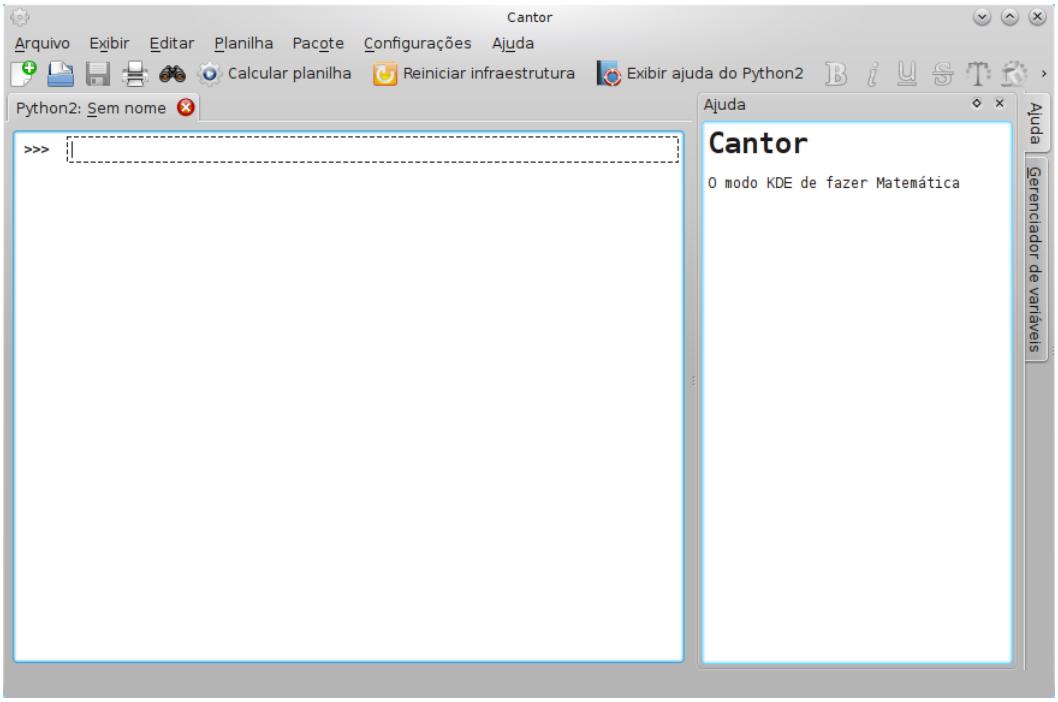

Figura 6. Tela inicial do Cantor

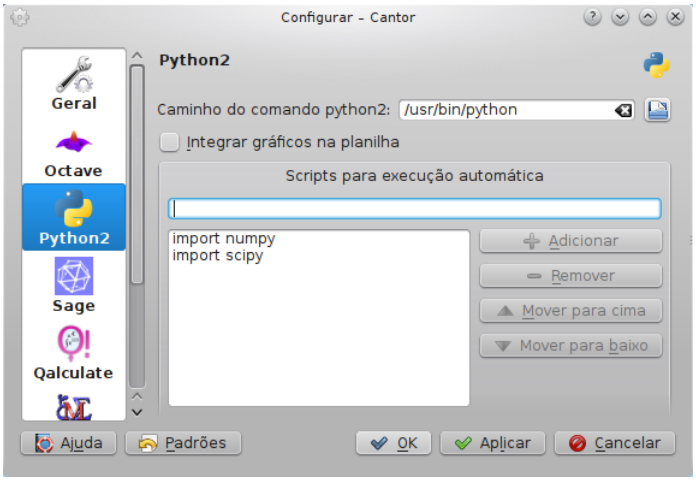

Figura 7. Tela de configuração do suporte a Python

às quais foram acrescentadas no painel *Gerenciamento de Variáveis*. Além destas, outras variáveis presentes no ambiente do interpretador Python estão também destacadas – como os pacotes carregados na inicialização do software.

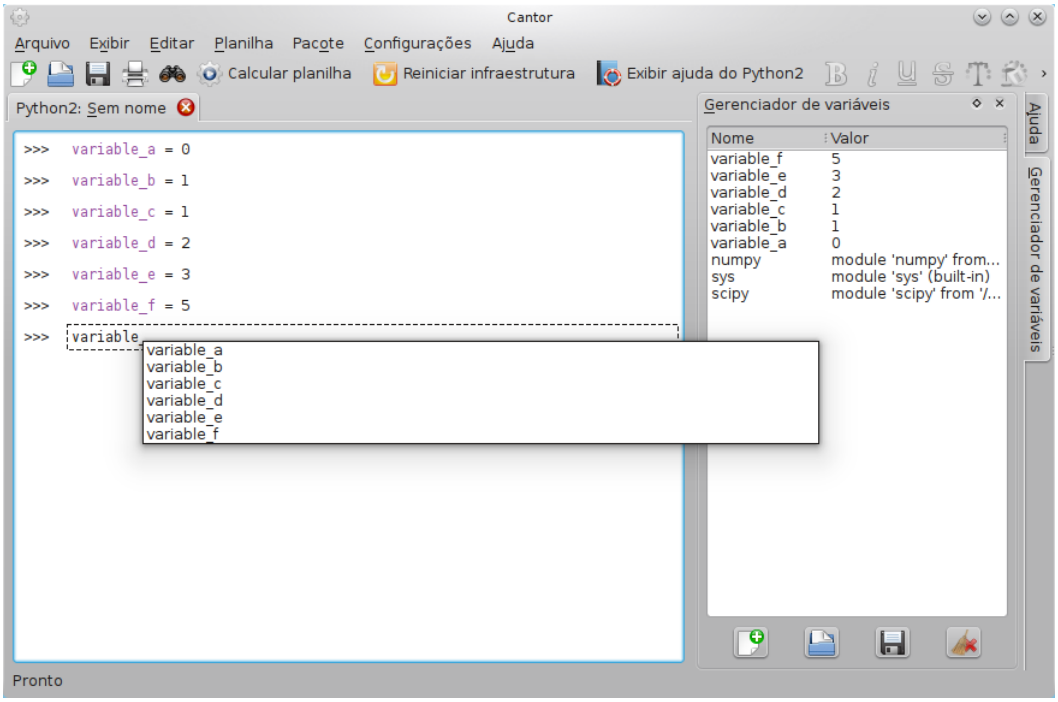

Figura 8. Destaque de sintaxe, complementação de código e gerenciamento de variáveis

Ainda sobre a Figura 8, é possível perceber a complementação de código quando o usuário digitou parte de um nome de variável. A complementação é ativada pressionando-se a tecla *TAB*, e também serve para os pacotes carregados no ambiente, fornecendo a complementação para funções presentes nestes.

O painel de gerenciamento de variáveis também permite acesso facilitado a funcionalidades de persistências de dados no Python. Na parte inferior do painel localizado no lado direito da Figura 8, há quatro botões responsáveis por esses acessos. São eles, da esquerda para a direita: *Adicionar uma nova variável*, *Carregar variáveis*, *Armazenar variáveis* e *Limpar variáveis*.

Simular-se-ão as três últimas funcionalidades descritas. Primeiro, clica-se no botão *Armazenar variáveis*, para salvar as variáveis que definimos durante essa utilização do software. O Cantor abrirá uma janela para definirmos o nome do arquivo que será usado para salvar as variáveis e onde queremos salvá-las. Em seguida, será carregado no terminal um *script* que irá iterar sobre as variáveis presentes no ambiente e salvá-las no arquivo. Esse *script* utiliza o pacote Python shelve, fornecido por padrão com a linguagem e responsável pela persistência de dados.

Em seguida, para excluir todas as variáveis do ambiente, pressiona-se o botão *Limpar variáveis*. Uma tela de confirmação será exibida e, após receber uma resposta positiva, um *script* responsável por iterar e apagar as variáveis será carregado no terminal. Tanto a operação de salvar as variáveis quanto a operação de limpar o ambiente podem ser

observadas na Figura 9. A operação logo após a definição da "variable_f" é a operação que salva as variáveis, enquanto a operação seguinte é aquela que apaga todas elas do ambiente. Percebe-se que o painel de gerenciamento de variáveis fica vazio após essa operação.

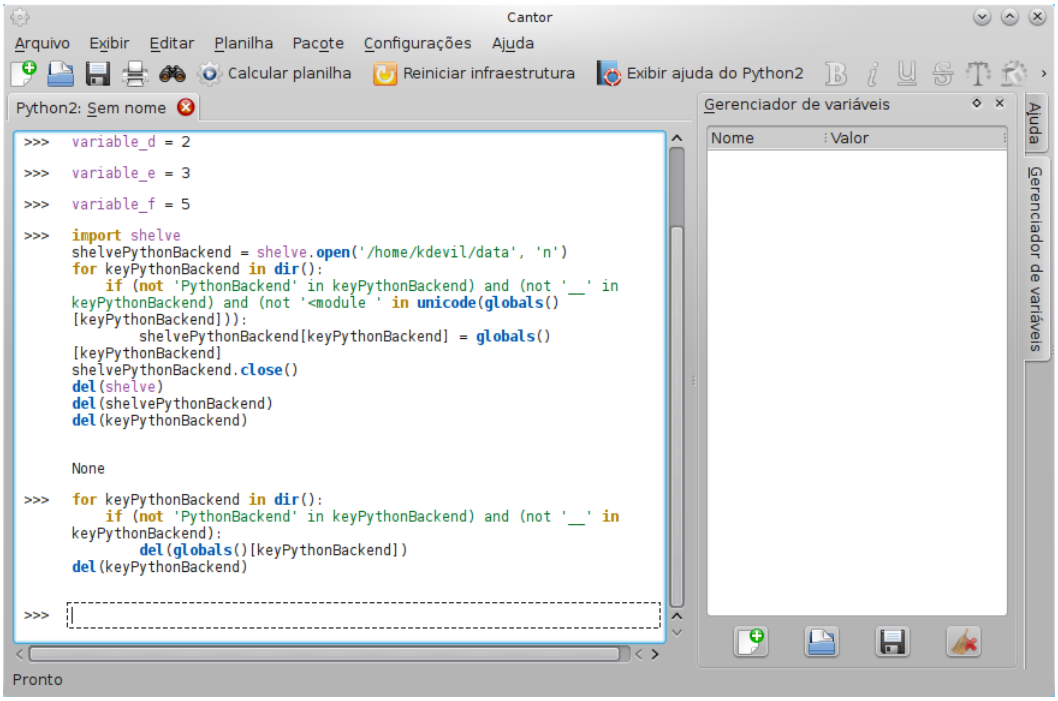

Figura 9. Salvando as variáveis e apagando-as do ambiente

Com o ambiente sem variáveis, é possível testar a funcionalidade de carregamento das variáveis salvas. Então, clicando-se no botão *Carregar variáveis*, surge uma janela para indicar que arquivo se quer carregar e, após isso, um novo *script* é carregado no terminal e as variáveis são lidas do arquivo e instanciadas no ambiente. Esse *script* também utiliza o módulo shelve para realizar a tarefa. A Figura 10 apresenta essa funcionalidade, com o *script* no terminal e o painel com o gerenciamento de variáveis atualizado.

Sobre o painel de ajuda, ele apresenta a saída do comando de ajuda ("help") do Python sem poluir o terminal. A Figura 11 apresenta o software quando é enviado o comando de ajuda para a função "complex" da biblioteca NumPy.

Clicando em *Exibir* $\Rightarrow$ *Mostrar o editor de script*, é possível acessar uma janela com um editor de *script* voltado para o desenvolvimento Python. Esse editor tem destaque de sintaxe e complementação de código funcionando, entretanto esta última ainda em fase experimental. Há funções para salvar o *script*, carregar, e mesmo executá-lo diretamente no terminal do Cantor, a partir do botão *Executar script*. A Figura 12 apresenta a saída do *script*, o mesmo da Figura 1, no terminal do Cantor.

Há ainda outras funcionalidades presentes no Cantor, entretanto, as apresentadas já permitem uma utilização simples e produtiva da linguagem Python para computação científica.

A sequir, a Seção 6 apresentará um exemplo de utilização do Cantor com Python

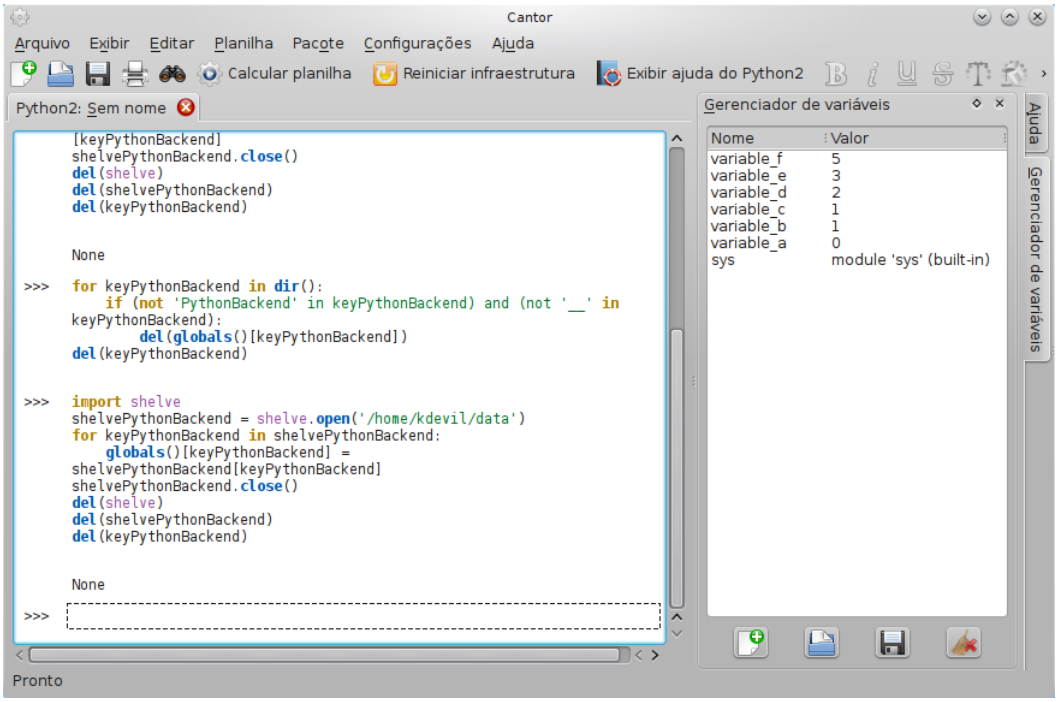

Figura 10. Carregando variáveis

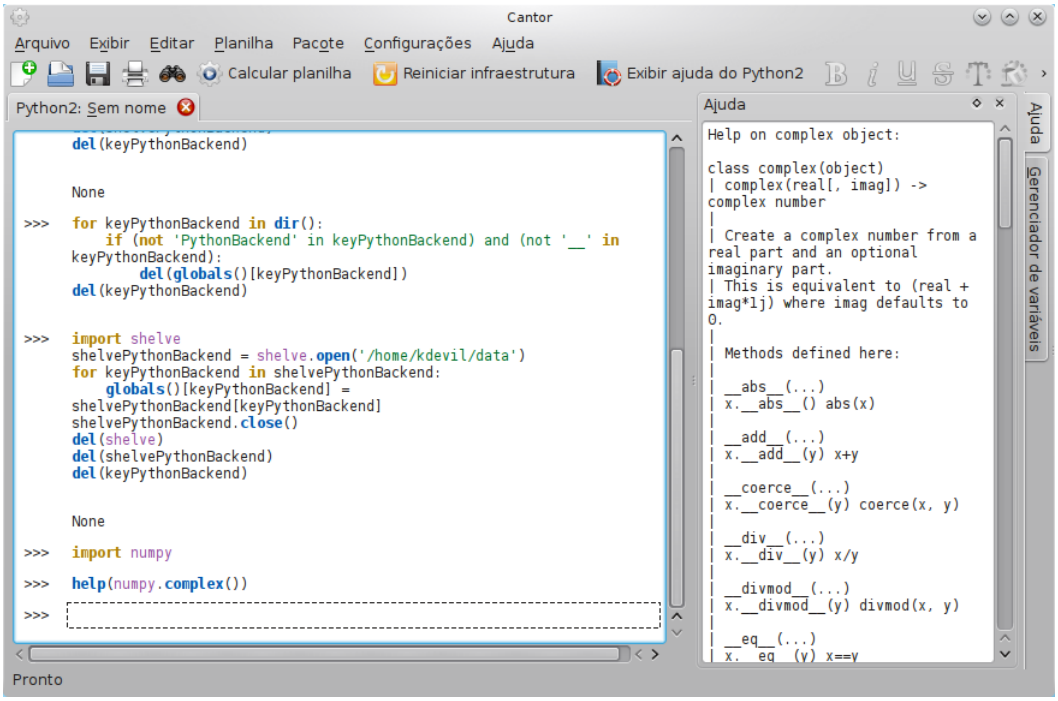

Figura 11. Painel de ajuda

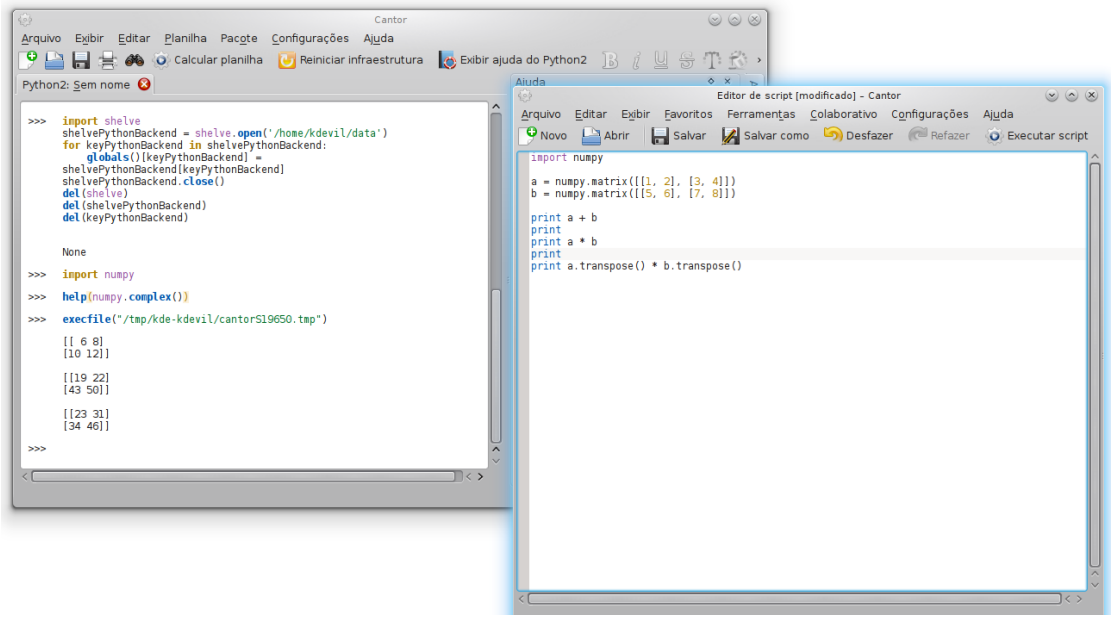

Figura 12. Editor de *script* do Cantor

para o desenvolvimento de redes neurais artificiais, bastante empregadas em diversos problemas típicos das áreas relacionadas com computação científica.

6. Exemplo de Utilização – Desenvolvimento de Redes Neurais Artificiais

Como uma ferramenta voltada para o auxílio a desenvolvedores, a apresentação de um exemplo de utilização do Cantor dificilmente poderá capturar por completo todas as possibilidades e usos que a ferramenta oferece àqueles que lidam nas diversas aplicações da computação científica; entretanto, ter um exemplo desses pode ter valor pedagógico ao mostrar a ferramenta em ação, mesmo que em uma aplicação não muito complexa.

Foi selecionada a tarefa de desenvolvimento de uma rede neural artificial como exemplo de utilização do Cantor. Redes neurais artificiais são poderosas ferramentas computacionais utilizadas em diversas áreas, das engenharias à matemática, física, computação e outras. Algumas de suas principais aplicações são classificação e reconhecimento de padrões, aproximação funcional, extração de regras, previsão, estimação, e mais [Silva et al. 2010].

O exemplo consistirá em criar três redes neurais artificiais do tipo perceptron e utilizá-las para aprender um conjunto de padrões formado por trinta elementos. Cada elemento desse conjunto contém três entradas e uma saída desejada.

A arquitetura das redes neurais terá duas camadas, a de entrada com três neurônios e a de saída com apenas um. O algoritmo de treinamento utilizado será o *backpropagation* e os erros de cada época do treinamento serão guardados para que, ao fim da fase de treinamento das três redes, possam ser gerados os gráficos do aprendizado (a curva de decrescimento dos erros) de cada uma. O conjunto de treinamento está disponível no Apêndice desse artigo.

Para realizar essa tarefa, serão utilizados os respectivos módulos do Python: numpy, para manipulação numérica de matrizes; pylab, para criação de gráficos; e o neu-

rolab[8], uma biblioteca de fácil utilização para a instanciação de redes neurais artificiais de diversos tipos. Como o foco do artigo é o Cantor, não serão aprofundadas as explicações sobre as funcionalidades dos módulos utilizados.

A Figura 13 apresenta a sequência de ações realizadas para cumprir o treinamento das redes neurais.

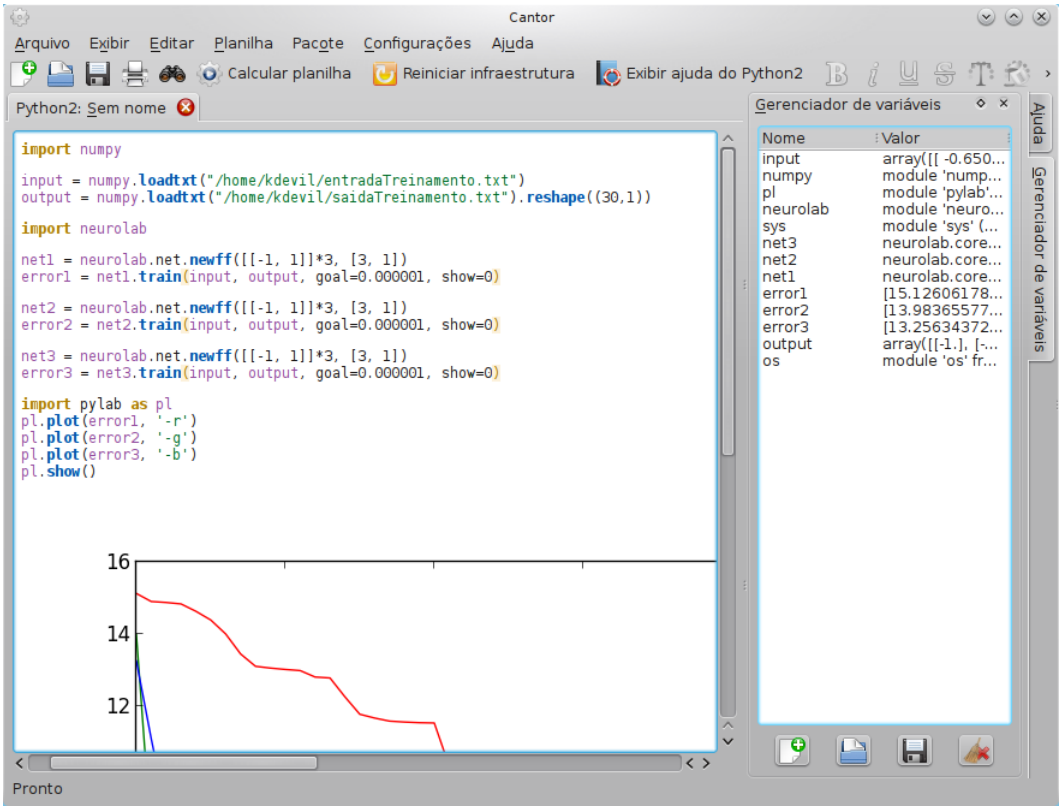

Figura 13. Exemplo de utilização – criação de redes neurais artificiais

O primeiro comando importa o módulo numpy para o ambiente do Python. Os dois comandos seguintes carregam os valores de entrada e saída de treinamento da rede a partir de arquivos texto instanciando matrizes para os mesmos.

Em seguida é importado o módulo de redes neurais neurolab. O próximo comando cria uma rede neural tipo perceptron e, logo a seguir, é realizado o treinamento da mesma, salvando-se os erros. Essa sequência de comandos é repetida mais duas vezes para criar e treinar mais duas redes neurais. Os nomes das redes são “net1”, “net2” e “net3”, e os erros obtidos no processo de treinamento são salvos, respectivamente, nos vetores “error1”, “error2” e “error3”.

No último bloco de comandos é importado o módulo para geração de gráficos pylab e são gerados os gráficos com os decréscimos dos erros obtidos durante o processo de treinamento das redes.

A Figura 14 apresenta esse gráfico, integrado ao ambiente de trabalho do Cantor.

[8]Neurolab está disponível em `http://code.google.com/p/neurolab/` (Acessado dia 28 de abril de 2014).

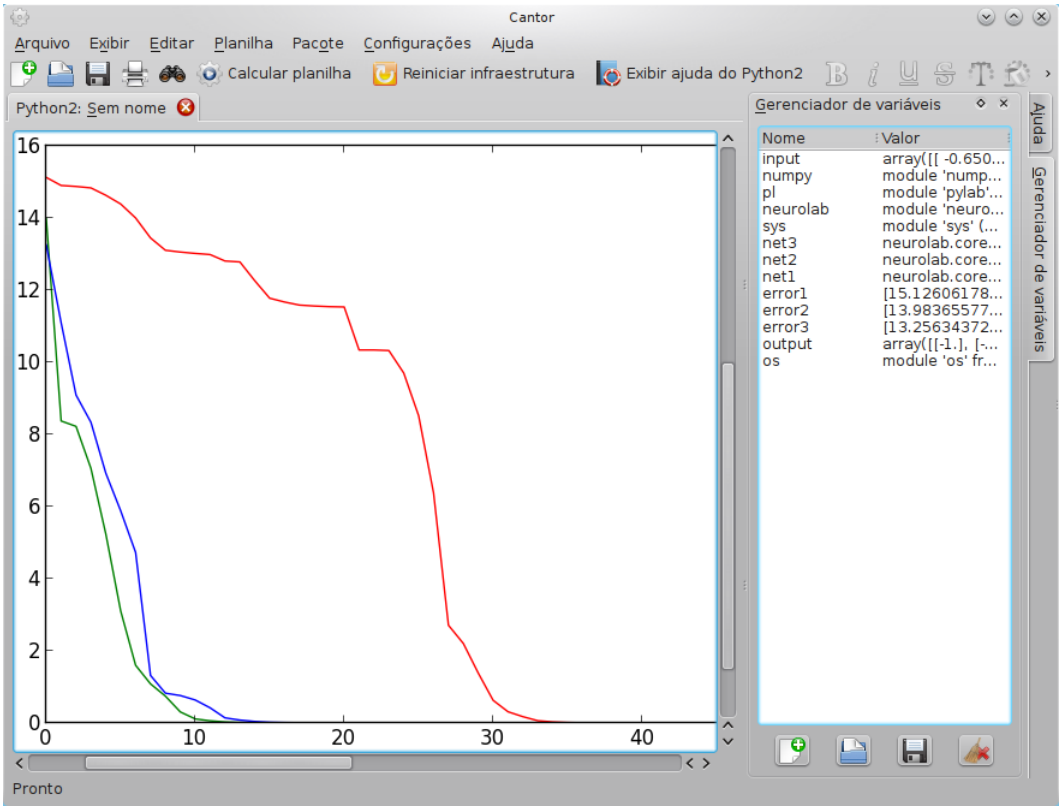

Figura 14. Gráfico com o decréscimo dos erros obtidos durante treinamento das redes neurais

Com esse exemplo foi demonstrado o uso do Cantor com Python, propiciando um ambiente com interessantes funcionalidades e facilidades para a tarefa de desenvolvimento de software nas áreas que trabalham com computação científica.

7. Conclusão e Trabalhos Futuros

Computação científica é uma importante área de aplicação e desenvolvimento de software, e pesquisadores, estudantes, e outros, de diferentes áreas, utilizam-na todos os dias para desempenhar suas atividades.

Python é uma linguagem de programação de caráter geral, mas devido à existência de diferentes pacotes voltados à computação científica, bem como a existência de conferências e uma comunidade específica sobre o tema, é crescente a adoção dessa linguagem nesse campo de estudos.

A disponibilidade de ambientes integrados de desenvolvimento, com diferentes funcionalidades que facilitam a utilização de linguagens como Python, auxiliam no crescimento do número de usuários e na familiarização do uso de linguagens como essas.

Apesar de haver uma plataforma proprietária dominante para esse tipo de aplicação, o desenvolvimento de softwares alternativos, principalmente livres, que entregam boas funcionalidades, podem se constituir como importantes ferramentas e serem opções viáveis para utilizadores que queiram migrar da solução proprietária.

Esse artigo descreveu o software Cantor, um ambiente voltado para a programação

científica, e o suporte à linguagem Python, desenvolvida pelo autor deste trabalho. Essa solução foi lançada em dezembro de 2013, junto com o pacote de aplicações providos pelo KDE em sua versão 4.12. Portanto, usuários GNU/Linux cujo KDE 4.12 esteja disponível nos repositórios de suas distribuições podem instalar o Cantor com suporte ao Python a partir destas distros, de maneira simplificada.

Em termos de desenvolvimento do software, há interessantes trabalhos futuros para essa plataforma, em especial o suporte à Python 3, o desenvolvimento de funcionalidades para gerenciamento de matrizes e gráficos utilizando *scripts* prontos, a criação de uma versão com suporte à Windows e OS X, entre outras. Por ser um software livre, todos os interessados estão convidados a contribuir com estas e outras funcionalidades.

Do ponto de vista acadêmico, os principais trabalhos futuros correspondem à estudos comparativos entre o Cantor utilizando Python e Matlab, principalmente em termos de funcionalidades quanto em verificação de usabilidade e aceitação de usuários. Também há a possibilidade de estudos quanto a aceitação dessa solução entre aqueles com motivações para migrar da plataforma proprietária, e também entre aqueles que já tem experiência no uso de Python em computação científica mas desconhecem o Cantor.

Agradecimentos

O desenvolvimento do suporte à linguagem de programação Python no Cantor foi parcialmente financiado pelo Google, via o programa Google Summer of Code de 2013. O autor é grato à empresa por este apoio.

O autor também presta seus agradecimentos ao desenvolvedor original do Cantor, Alexander Rieder, pela tutoria durante a realização desse projeto e por toda orientação gentilmente oferecida desde o início.

Também fica registrado o agradecimento à comunidade KDE, que deu todo o suporte e proveu ferramentas fundamentais para o desenvolvimento desse projeto.

Referências

Gomez, C. (1999). *Engineering and scientific computing with Scilab*. Springer.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95.

Inc., T. M. (2010). *Matlab: software for numerical computation*. The MathWorks Inc., Natick, Massachusetts.

Michael, T. H. (2002). *Scientific computing: an introductory survey*. The McGraw-Hill Companies Inc., New York.

Moler, C. B. (2008). *Numerical computing with MATLAB*. Siam.

Oliphant, T. E. (2007). Python for scientific computing. *Computing In Science & Engineering*, 9(3):10–20.

Pérez, F. and Granger, B. E. (2007). IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29.

Prabhu, P., Jablin, T. B., Raman, A., Zhang, Y., Huang, J., Kim, H., Johnson, N. P., Liu, F., Ghosh, S., Beard, S., et al. (2011). A survey of the practice of computational science. In *State of the Practice Reports*, page 19. ACM.

Scilab Enterprises (2012). *Scilab: Free and Open Source software for numerical computation*. Scilab Enterprises, Orsay, France.

Silva, I. N. d., Spatti, D. H., and Flauzino, R. A. (2010). *Redes neurais artificiais para engenharia e ciências aplicadas*. Artliber, São Paulo.

Apêndices

Conjunto de Treinamento das Redes Neurais Artificiais

Segue abaixo a Tabela 7 com os dados do conjunto de treinamento utilizado nas redes neurais artificiais desenvolvidas no artigo.

Tabela 1. Tabela com dados de treinamento das redes neurais

Entrada 1	Entrada 2	Entrada 3	Saída
-0.6508	0.1097	4.0009	-1.0000
-1.4492	0.8896	4.4005	-1.0000
2.0850	0.6876	12.0710	-1.0000
0.2626	1.1476	7.7985	1.0000
0.6418	1.0234	7.0427	1.0000
0.2569	0.6730	8.3265	-1.0000
1.1155	0.6043	7.4446	1.0000
0.0914	0.3399	7.0677	-1.0000
0.0121	0.5256	4.6316	1.0000
-0.0429	0.4660	5.4323	1.0000
0.4340	0.6870	8.2287	-1.0000
0.2735	1.0287	7.1934	1.0000
0.4839	0.4851	7.4850	-1.0000
0.4089	-0.1267	5.5019	-1.0000
1.4391	0.1614	8.5843	-1.0000
-0.9115	-0.1973	2.1962	-1.0000
0.3654	1.0475	7.4858	1.0000
0.2144	0.7515	7.1699	1.0000
0.2013	1.0014	6.5489	1.0000
0.6483	0.2183	5.8991	1.0000
-0.1147	0.2242	7.2435	-1.0000
-0.7970	0.8795	3.8762	1.0000
-1.0625	0.6366	2.4707	1.0000
0.5307	0.1285	5.6883	1.0000
-1.2200	0.7777	1.7252	1.0000
0.3957	0.1076	5.6623	-1.0000
-0.1013	0.5989	7.1812	-1.0000
2.4482	0.9455	11.2095	1.0000
2.0149	0.6192	10.9263	-1.0000
0.2012	0.2611	5.4631	1.0000