Computing (2013) 95 (Suppl 1):S675–S694
DOI 10.1007/s00607-012-0255-3

NCLab: Public Computing Laboratory

Sascha M. Schnepp

Received: 20 November 2012 / Accepted: 3 December 2012 / Published online: 1 January 2013

Abstract This survey paper describes the Network Computing Laboratory (NCLab), a novel public cloud computing platform for mathematics, programming, scientific computing and computer simulations. Through a web-browser interface, it provides users with free access to interactive graphical modules that include symbolic and numerical methods, programming in several languages, computing with Python scientific libraries, computing with GNU Octave, GPU computing with CUDA, computational geometry, 3D CAD design, computational graph theory, finite element programming with the Hermes library, and interactive graphical finite element modules. Users can upload files and data from their local computers, clone projects from the database, share files, form teams, and collaborate on projects. This paper briefly describes how NCLab operates, and it provides concise descriptions of NCLab computational modules with examples of use.

Keywords Scientific computing · Network computing · Cloud computing · GPU computing · Numerical methods · Python · Scipy · Octave · CUDA · Finite element method

Mathematics Subject Classification 65F05 · 65F10 · 65T50 · 65L60 · 65L05 · 68W30 · 05C85 · 97N30 · 97N40 · 97N80

1 Basic terminology of cloud computing

By *cloud computing* one usually means the use of computing resources (hardware and software) that are delivered as a service over the Internet [1]. The word *cloud* in this

S. M. Schnepp (✉)
Laboratory for Electromagnetic Fields and Microwave Electronics,
ETH Zürich, Gloriastrasse 35, 8092 Zürich, Switzerland
e-mail: schnepps@ethz.ch

context stands for a (typically large) computer cluster where virtual computers with variable parameters (disk space, runtime memory, number of cores) can be allocated dynamically based on actual demand. This model is based on "renting" rather than "owning" and it has some obvious advantages—for example, the user does not have to maintain or renew the hardware. The rapid expansion of this new paradigm can be illustrated by a large amount of new terminology that appeared in the last few years:

- *Infrastructure as a service (IaaS)* This is the most basic cloud computing model where providers offer access to their servers. The user typically has admin access to the server and can install and run his own software.
- *Platform as a service (PaaS)* This model provides a comprehensive computing platform that typically includes tools to develop software, database, web server etc.
- *Software as a service (SaaS)* In this model the provider installs and maintains some software on his servers and lets users access and use the software over the Internet.
- *Storage as a service (STaaS)* Providers offer storage space that typically is more cost-effective than if the user stored the data on his hardware.
- *Security as a service (SECaaS)* Providers offer access to their security services including authentication, anti-virus, anti-malware or spyware, intrusion detection, security event management and others.
- *Data as a service (DaaS)* Provider offers data storage with additional services to ensure security and fast access to the data over the Internet.
- *Desktop virtualization* This concept separates a personal computer desktop environment from a physical machine using the client-server model. While the operating system and applications run on a server, the virtual image is transferred to the client (user's computer) in real time. This model requires substantial bandwidth.

2 Network Computing Laboratory (NCLab)

Technically, NCLab falls into the SaaS category. It does not use desktop virtualization. While the server-side is running on a Linux server, the user interface works natively in the web browser on the client and to some extent it resembles Windows. It will be called "Desktop" in the following, which should not be confused with the desktop of the computer or laptop that is used to access NCLab. In principle, any device that has a web browser can be used to access NCLab. Currently, NCLab has not been optimized for very small displays though, so accessing it from smart phones is not recommended. NCLab has been partially optimized for touch interfaces (tablets).

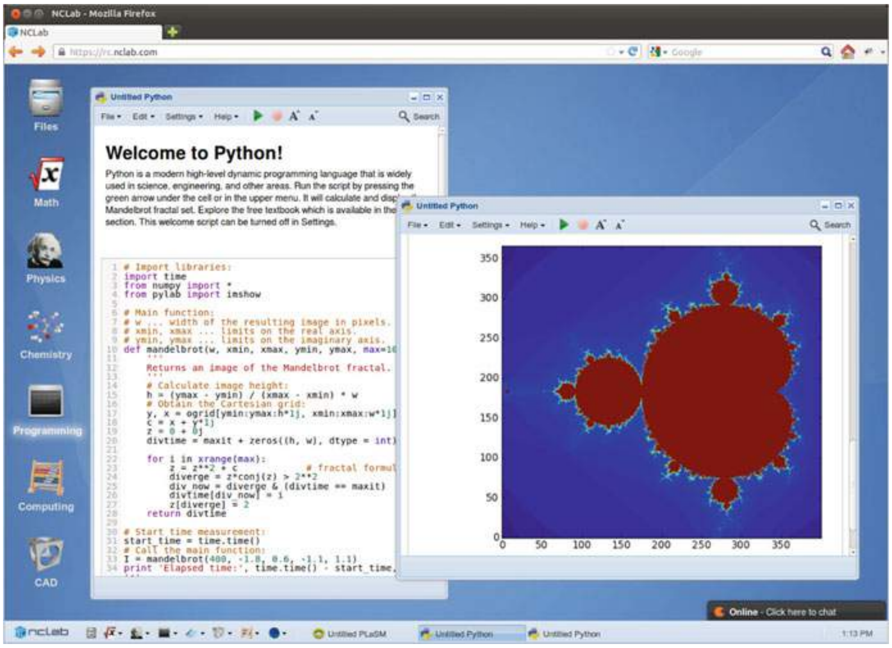
User data is stored on the server using MongoDB, a scalable, high-performance, open source NoSQL database [2]. The Desktop provides a File Manager that can access the database and display the user data on the client as files and folders that resemble a usual file system. NCLab uses a leading commercial cloud provider [3] and part of the service includes regular database backups. The security of user data is fully handled by the cloud service provider. Secure EV SSL encrypted communication is used for sending data to the server and back.

NCLab modules and applications are installed on the server. They include a combination of proprietary software developed by FEMhub Inc. and open source projects related to scientific computing whose license permits server-side use (i.e., LGPL or more permissive). NCLab provides access to a limited number of GPL-licensed softwares such as Octave, but in this case the software is not linked to the server-side of NCLab and instead it is used in standalone batch mode.

3 Python programming

Next let us illustrate the user's workflow in NCLab using Python programming as an example. Python [20] is a modern high-level dynamic programming language that is used frequently in modern scientific computing applications. A free Python programming textbook is provided to NCLab users at [4]. Currently, NCLab provides Python 2.7. After creating a free account and logging in, the user sees a desktop that resembles a regular computer desktop.

After double-clicking (or tapping) on the Programming icon, a menu of programming languages appears that contains Python. Clicking on Python launches a Python worksheet. Initially, the worksheet contains a demo script (that can be turned off in Settings). The demo script can be evaluated by pressing a green arrow button. This is illustrated in Fig. 1.



Welcome to Python!

Python is a modern high-level dynamic programming language that is widely used in science, engineering, and other areas. Run the script by pressing the green arrow under the cell or in the upper menu. It will calculate and disp Mandelbrot fractal set. Explore the free textbook which is available in the section. This welcome script can be turned off in Settings.

```
# Import libraries:
import time
from numpy import *
from pylab import imshow

# Main function:
# w ... width of the resulting image in pixels.
# xmin, xmax ... limits on the real axis.
# ymin, ymax ... limits on the imaginary axis.
def mandelbrot(w, xmin, xmax, ymin, ymax, max=1
    '''
    Returns an image of the Mandelbrot fractal.
    '''
    # Calculate image height:
    h = (ymax - ymin) / (xmax - xmin) * w
    # Obtain the Cartesian grid:
    y, x = ogrid[ymin:ymax:h*1j, xmin:xmax:w*1j]
    c = x + y*1j
    z = 0 + 0j
    divtime = maxit + zeros((h, w), dtype = int)

    for i in xrange(max):
        z = z**2 + c                  # fractal formu
        diverge = z*conj(z) > 2**2
        div_now = diverge & (divtime == maxit)
        divtime[div_now] = i
        z[diverge] = 2
    return divtime

# Start time measurement:
start_time = time.time()
# Call the main function:
I = mandelbrot(400, -1.8, 0.6, -1.1, 1.1)
print 'Elapsed time:', time.time() - start_time,
```

Fig. 1 NCLab desktop with a Python worksheet that constructs the Mandelbrot fractal

Pressing the green arrow sends the script to the server. The server assesses the availability of computing nodes, selects a compute node with largest available computing resources, and starts a Python engine on the node. The word *engine* is standard in the SaaS context, and in this concrete situation it means an instance of the Python interpreter that runs in a secure environment that is isolated from other processes running on the node. Hence, one user cannot interfere with processes of any other user or temper with files other than his own.

After the Python interpreter finishes, the output that may include text and/or images is sent to the server and from there to the client. The Python engine is kept alive for some time so that the user does not lose his variables and intermediate results, but it is terminated when it has been idle for too long or when the user closes the Python worksheet. The output of the concrete demo script shown above is an image of the Mandelbrot fractal.

The Python worksheet shown in Fig. 1 contains one descriptive HTML cell and one input code cell. In reality there can be more of each type and the user is free to edit them, reorder them, and perform various other operations that we will not discuss here. Perhaps the one thing worth mentioning is that pressing the green button in the upper menu will evaluate all code cells in the worksheet. Each code cell also has its own green arrow button located below it on the left-hand side, that can be used to evaluate just this particular cell. Programs created in NCLab can be saved using the File menu of the worksheet, and the file manager can also be used to upload Python files from the user's computer.

Since this paper is not meant to substitute the NCLab user's manual, we will not discuss other technical details related to working in NCLab. Such information can be found via links provided on NCLab's home page or using Help sections of particular applications. Similarly, we will not discuss Python programming in any more detail. The important information is that Python programs can be composed in NCLab, or they can be uploaded and run on a remote server.

In the following we will describe from the user's perspective modules that are related to scientific computing and computer simulations.

4 Computing with Python scientific libraries

NCLab provides a comprehensive list of Python scientific computing libraries including Scipy [5], Numpy [6], Pylab [7], Matplotlib [8], Sympy [9] and others. Since not all readers may be familiar with them, let us describe them briefly:

Scipy is an open source library of algorithms and mathematical tools for the Python programming language. It contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering. It has a similar audience to applications such as Matlab, GNU Octave, and Scilab. For illustration of how Scipy is used, let us give a few examples related to image manipulation:

```
# Import libraries
import pylab as pl
from scipy import ndimage, misc

# Import the Lena image from Scipy.misc:
L = misc.lena()

# Clear screen:
pl.clf()

# Display the image:
im = pl.imshow(L, cmap='hot')
pl.show()

# Cropping:
lx, ly = L.shape
crop_lena = L[lx/4:-lx/4, ly/4:-ly/4]
pl.imshow(crop_lena, cmap='hot')
pl.show()

# Vertical flip:
flip_ud_lena = flipud(L)
pl.imshow(flip_ud_lena, cmap='hot')
pl.show()

# Rotation with reshape:
rotate_lena = ndimage.rotate(L, 45)
pl.imshow(rotate_lena, cmap='hot')
pl.show()

# Blurring:
blurred_lena = ndimage.gaussian_filter(L, sigma=5)
pl.imshow(blurred_lena, cmap='hot')
pl.show()

# Sharpening:
filter_blurred_l = ndimage.gaussian_filter(blurred_lena, 1)
alpha = 30
sharpened_lena = blurred_lena + alpha *
                 (blurred_lena - filter_blurred_l)
pl.imshow(sharpened_lena, cmap='hot')
pl.show()
```

This script has several outputs which are shown in Figs. 2, 3, 4.

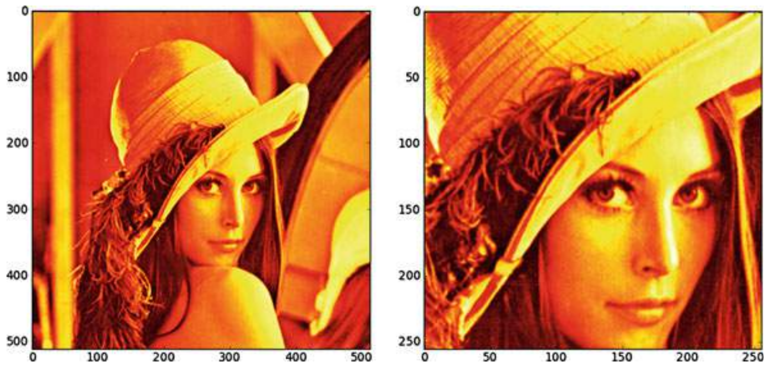


Fig. 2 *Left* Original Lena 512 × 512 pixels. *Right* Cutting off 25 % from each side

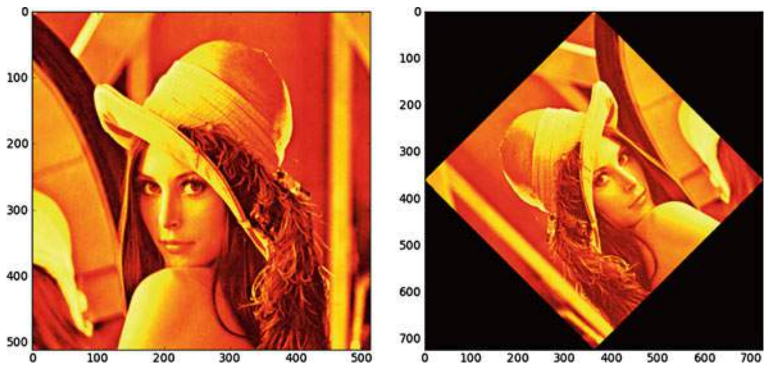


Fig. 3 *Left* Flipped from left to right. *Right* 45° rotation with reshape

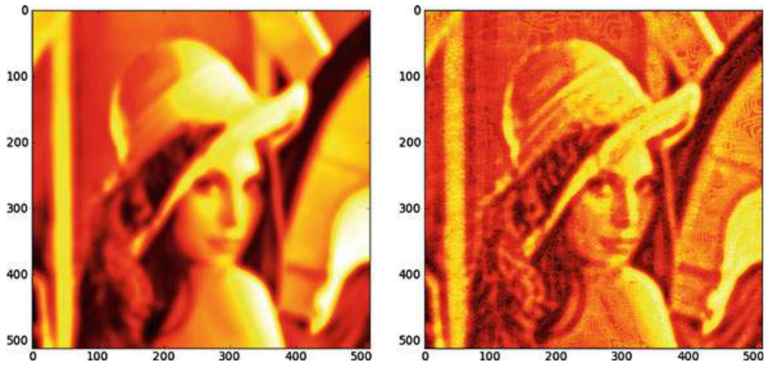


Fig. 4 *Left* Blurring via a Gaussian filter. *Right* Sharpening

Numpy is the fundamental package for scientific computing with Python. It contains among others a powerful N-dimensional array object, sophisticated (broadcasting) functions, tools for integrating C/C++ and Fortran code, linear algebra, Fourier transform, and random number capabilities.

The usage of Numpy can be illustrated using a script that performs the fast Fourier transform (FFT) of a highly oscillatory function

$$y = sin(200x)$$

that is perturbed using random fluctuations:

```
# Import libraries:
from random import random
from numpy import linspace, pi, sin
from pylab import plot, clf, legend, show

# Define a sine function in (0, 2*pi)
x = linspace(0, 2 * pi, 1000)
y = sin(200*x)

# Small random perturbation:
n = len(x)
for i in range(n):
  y[i] += random() * 0.2

# Plot y(x):
clf()
plot(x, y, label="y(x)")
legend()
show()

# FFT y(x):
from scipy.fftpack import fft
y_transformed = fft(y)

# Plot y_transformed(x):
clf()
plot(x, y_transformed, label="y_transformed(x)")
legend()
show()
```

The function $y(x)$ is shown in Fig. 5.

The result after performing FFT is shown in Fig. 6.

Pylab is a combination of Python, Numpy, Scipy, Matplotlib, and IPython that provides a compelling environment for numerical analysis and computation.

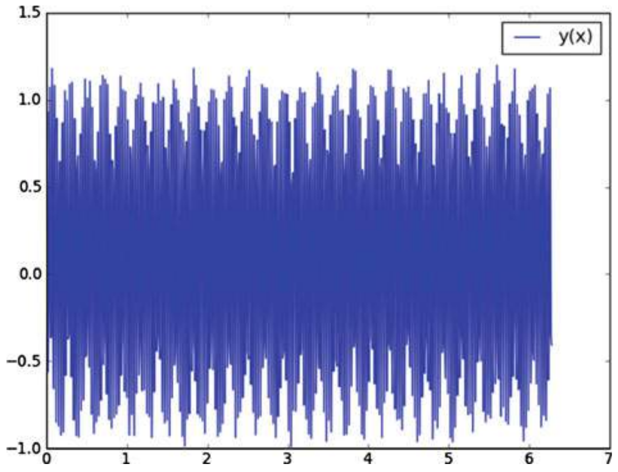


Fig. 5 Original function $y(x)$

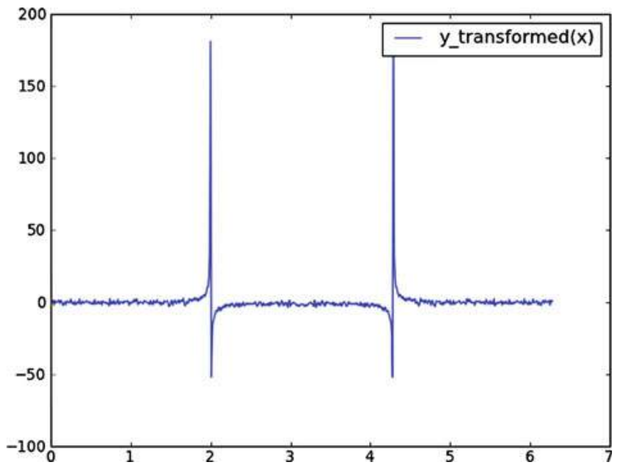


Fig. 6 Fourier transform of the function $y(x)$

Matplotlib is the major Python plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.

Last, Sympy is an open source Python library for symbolic mathematics that aims to become a full-featured computer algebra system (CAS). The usage of Sympy can be illustrated by solving symbolically a nonlinear ordinary differential equation

$$(x^2 - 9)y' + yx = x^2$$

The corresponding script has the form

```
# Solve the equation:
A = dsolve((x**2 - 9)*diff(y(x), x) + y(x) * x - x**2, y(x))

# Pretty print the result:
pprint(A)
```

The output is

```
                          ________            /x\
                         /  2         9*acosh|-|
                    x*\/   x   - 9           \3/
               C1 + ------------- + ----------
                          2              2
        y(x) = ------------------------------
                              ________
                             /  2
                           \/  x   - 9
```

5 Computing with GNU Octave

GNU Octave [10] is a high-level programming language for scientific computing whose syntax is very similar to Matlab. Most Matlab m-files do not require any changes to run in Octave. The Octave module is part of the Computing module and its usage is analogous to how Python is used. Users have three ways to work with Octave files in NCLab:

- Upload existing Matlab/Octave files from their local hard disk.
- Clone Matlab/Octave files from the database of public projects.
- Compose Octave programs from scratch using the Octave worksheet.

The uploaded or cloned files can be further edited in the Octave worksheet. For illustration, let us show a simple Octave/Matlab script that calculates an approximation of the number π by inserting a large number of random points into the unit square and counting how many also lie in the unit circle:

```
function y = stochastic_pi(n)
    x = rand(n, 1);
    y = rand(n, 1);
    y = 4 * (sum( 1 - sign(sqrt(x.^2 + y.^2) - 1)) / 2) / n;
endfunction

% Test case:
n = 10000
stochastic_pi(n)
```

Sample output of this script is

```
n =  10000
ans =  3.1408
```

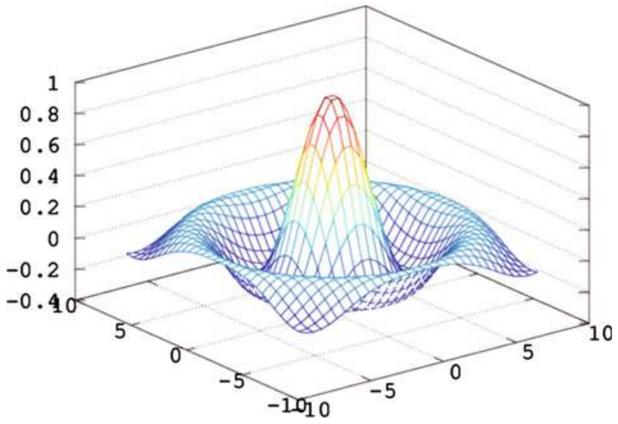


Fig. 7 Output of a demo script displaying a real function of two variables

In order to illustrate the plotting capabilities, let us evaluate another simple Octave/Matlab script that displays a function of two variables:

```
tx = ty = linspace (-8, 8, 30)';
[xx, yy] = meshgrid (tx, ty);
r = sqrt (xx .^ 2 + yy .^ 2);
tz = sin (r) ./ r;
mesh (tx, ty, tz);
```

The output is shown in Fig. 7.

The NCLab database of public projects, accessible through the file manager's Project menu or through the Octave worksheet's File menu, contains many Octave programs for numerical methods courses including the Taylor polynomial, rootfinding, interpolation, approximation, numerical quadrature, solution of systems of linear and nonlinear algebraic equations, linear and nonlinear regression, solution of ODE and PDE, etc. Every user can add their own programs.

6 GPU computing with PyCUDA

GPU computing or GPGPU (General Purpose GPU) computing is a computing paradigm that combines the use of a GPU (Graphics Processing Unit) with the use of a CPU to accelerate general-purpose scientific and engineering applications. The development of GPU computing was pioneered around 2,000 by Nvidia. Whereas a number of consumer graphics cards suitable for small to middle-sized problems are available at very reasonable prices, graphics cards that are powerful enough for advanced scientific computations are still relatively costly (in the order of thousands of dollars).

NCLab provides several GPU units that can be accessed freely from the PyCUDA worksheet. Here PyCUDA [11] stands for Python wrappers for CUDA, Nvidia's programming language for GPU units. PyCUDA was created by Andreas Klöckner around 2009, and since then it was expanded with the help of many contributors.

The database of public projects in NCLab contains around 30 tutorial projects that every user can clone into his/her account and run instantly. These projects are based on the official PyCUDA tutorial [11] and they were included in NCLab with the author's consent.

7 Computational graph theory with the NetworkX library

NetworkX [12] is a Python-based software package for the creation, manipulation, and study of the structure, dynamics, and functions of graphs and complex networks, developed by Aric Hagberg [13] at the Los Alamos National Laboratory. Tutorial and many examples can be found on the project home page. Here, let us illustrate its usage on a rather simple example.

We generate a random graph with 1,000 nodes and 5,000 edges, and calculate the eigenvalues of the generalized Laplacian:

```
# Import libraries:
import numpy.linalg
eigenvalues = numpy.linalg.eigvals
from pylab import *

# Define number of nodes and edges:
n = 1000
m = 5000

# Construct random graph:
G = gnm_random_graph(n, m)

# Create the generalized Laplacian:
L = generalized_laplacian(G)

# Calculate eigenvalues:
e = eigenvalues(L)
print "Largest eigenvalue:", max(e)
print "Smallest eigenvalue:", min(e)

# Histogram with 100 bins:
hist(e,bins = 100)

# Eigenvalues between 0 and 2:
xlim(0, 2)

# Display the graph:
show()
```

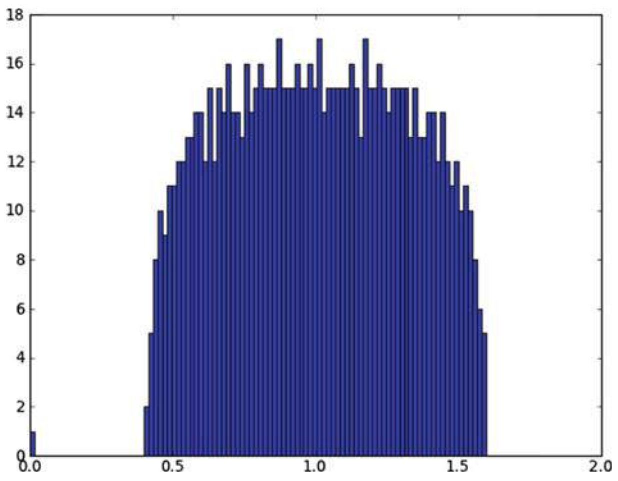


Fig. 8 Calculation of eigenvalues of the generalized Laplacian of a random graph with 1,000 nodes and 5,000 edges

The output of this script has two parts. First the text:

```
Largest eigenvalue: 1.5981610973873812
Smallest eigenvalue: -1.0896791983286371e-16
```

Then the graphics (Fig. 8):

8 Computational geometry with the PLaSM library

PLaSM [14] stands for *Programming Language of Solid Modeling*. This simple and elegant scripting language along with an open source collection of powerful multi-dimensional computational geometry algorithms behind it was created by A. Paoluzzi et al. [15] at the University of Rome in Italy. It allows the user to create many types of simple objects, transform them using scaling, rotations and translations, perform intersections and unions of objects, subtract objects from each other, and define many types of curved surfaces. A free open source textbook on PLaSM is available as well [16].

The usage of PLaSM can be illustrated on the following sample script that creates a cube of size 2, and gradually subtracts from it three cylinders of radius 0.75 and height 4 that are first translated and rotated into the three main axial directions:

```
# Create a cube of size 2:
c = CUBE(2)

# Create a cylinder of radius 0.75 and height 4:
cyl = CYLINDER(0.75, 4)

# Translate the cube by -1 in each axial direction:
c = T(c, -1, -1, -1)

# Translate the cylinder by -2 in the z-direction:
cyl = T(cyl, 0, 0, -2)

# Subtract cylinder from the cube:
c = DIFF(c, cyl)

# Rotate the cylinder by 90 degrees about the x-axis:
cyl = R(cyl, 1, PI/2)

# Subtract the cylinder from the cube:
c = DIFF(c, cyl)

# Rotate the cylinder by 90 degrees about the z-axis:
cyl = R(cyl, 3, PI/2)

# Subtract the cylinder from the cube:
c = DIFF(c, cyl)

# View the result:
lab.view(c)
```

After understanding this minimum, the reader is already capable of creating geometries that do not include advanced curved surfaces such as splines or Bézier curves. The commands are fairly self-explanatory but let us add a few comments. The command:

```
c = CUBE(2)
```

creates a cube `c` of edge length 2. PLaSM uses symbols for objects in order to allow performing operations with them. In fact, these are variables as one knows them from computer programming. Command

```
c = T(c, x, y, z)
```

translates object `c` by a 3D vector `(x, y, z)`. Command

```
cyl = R(cyl, m, a)
```

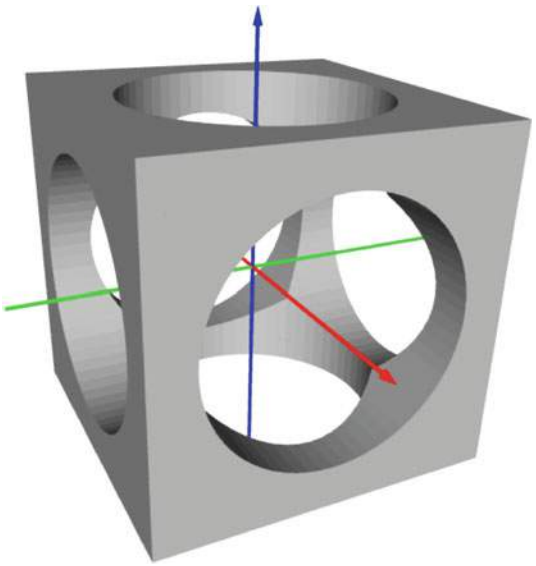


Fig. 9 Cube after subtracting three cylinders in the main axial directions

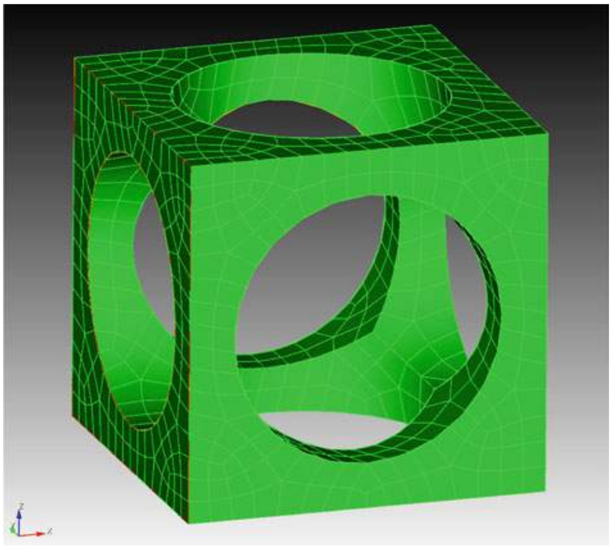


Fig. 10 3D mesh created by CUBIT using the STL file corresponding to Fig. 9

rotates object `cyl` about the `m`-th axis by angle `a`. Here `m = 1` means the x-axis etc. Last, command

```
c = DIFF(c, cyl)
```

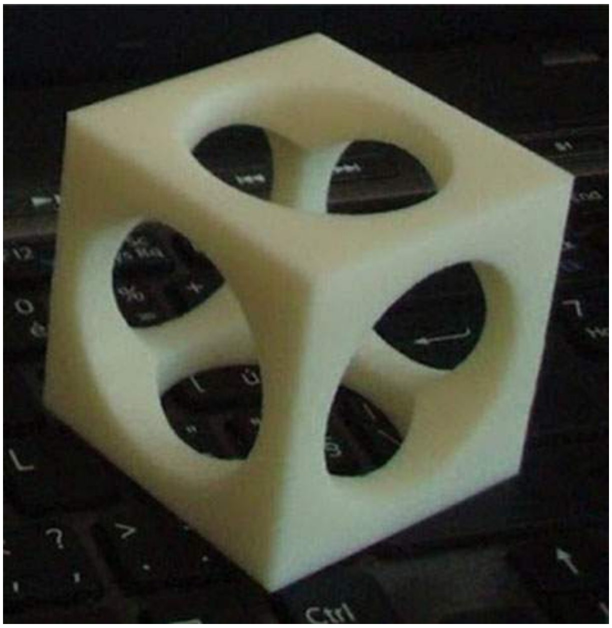


Fig. 11 3D print of the geometry shown in Fig. 9

subtracts object `cyl` from object `c`. Of course one can use arbitrary names to denote objects and the resulting object could be called differently from `c`. PLaSM also provides other standard binary operations `INTERSECTION`, `UNION` and `XOR`.

The output of the above script is shown in Fig. 9.

NCLab makes it possible to export surface triangulations as STL files. These can be imported into mainstream mesh generation packages and used for surface and volumetric mesh generation. Figure 10 shows the corresponding hexahedral mesh generated using CUBIT.

STL files are also accepted by most 3D printers. Figure 11 presents such a print corresponding to Fig. 9.

9 Solving PDE with the Hermes library

Hermes (**H**igher-ord**er** **m**odular finite **e**lement **s**ystem) [17] based on [18] is an open source C++ library for rapid development of adaptive hp-FEM / hp-DG solvers. Novel hp-adaptivity algorithms help solve a large variety of problems ranging from ODE and stationary linear PDE to complex time-dependent nonlinear multiphysics PDE systems.

Hermes is used in NCLab via its Python wrappers. The usage of the wrappers is straightforward as there are only a few naming conventions that relate the names of C++ classes, functions, and variables to their Python counterparts. About a dozen

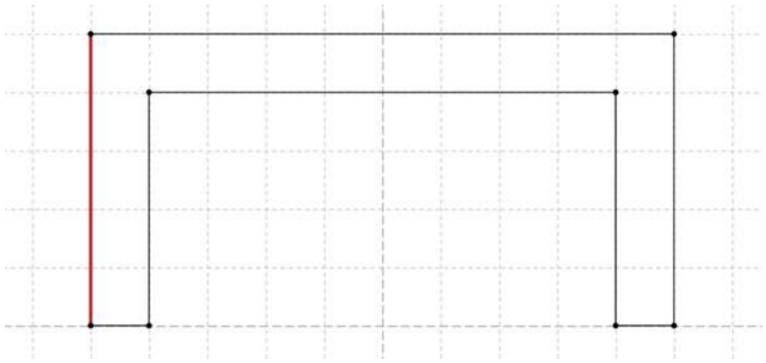


Fig. 12 Frame geometry

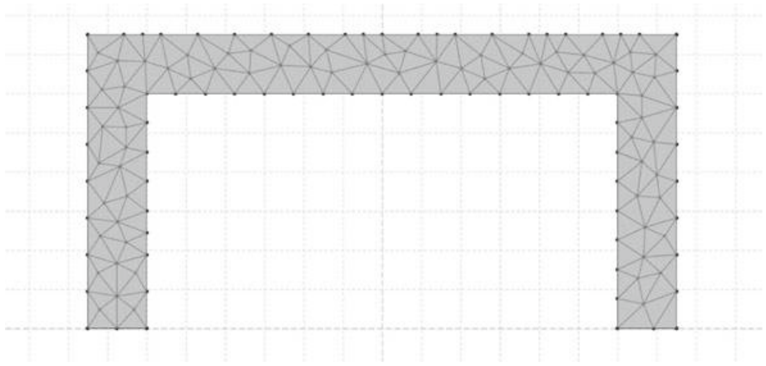


Fig. 13 Mesh generated using triangle

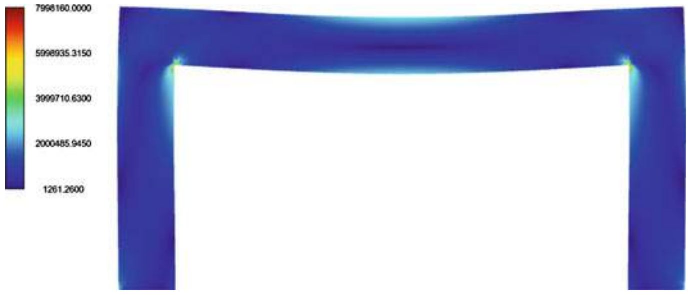


Fig. 14 Von Mises stresses (displacement is magnified for visualization purposes)

tutorial examples can be be cloned from the database. In Appendix A at the end of this article we show a sample program that employs Hermes to define equations of linear elasticity and solve an example problem.

10 Interactive graphical FEM modules

Recently, three entry-level interactive graphical modules have been introduced into NCLab. They can deal with problems in electrostatics with fixed potential (Dirichlet) and charge density (Neumann) conditions, linear elasticity with fixed displacement (Dirichlet) and surface force (Neumann) conditions, and general linear second-order equations of the form

$$-\sum_{i,j=1}^{2} \frac{\partial}{\partial x_i}\left(a_{ij}\frac{\partial u}{\partial x_j}\right) + \sum_{i=1}^{2} b_i \frac{\partial u}{\partial x_i} + cu = f$$

with constant coefficients and Dirichlet, Neumann and Newton boundary conditions. These modules employ higher-order finite elements and curvilinear elements, but they are not capable of automatic adaptivity yet.

For illustration, let us use the linear elasticity module to calculate displacements and stresses in a 2D steel frame of outer measures $a \times b$ where $a = 1$ m and $b = 0.5$ m, and thickness $h = 0.1$ m. The frame is fixed on the two bottom edges and loaded with a vertical force $F = 10^5$ N on the top edge. The geometry shown in Fig. 12 was created using the interactive geometry editor of the linear elasticity module.

Part of the geometry definition is the assignment of markers for boundary edges where boundary conditions will be prescribed. After the geometry is finished, it is sent to the server for mesh generation. The mesh generated using Triangle [19] is shown in Fig. 13.

Last, after selecting equation parameters and associate concrete boundary conditions with the previously defined edge markers, one sends the data to the server for processing. The mesh shown in Fig. 13 was twice uniformly refined and equipped with quadratic elements. The number of degrees of freedom (DOF) for this computation was 39,774. The resulting Von Mises stresses in the structure are shown in Fig. 14.

11 Sample Hermes program: linear elasticity

The following program uses the Hermes library via its Python wrappers in NCLab to define weak forms for the Lamé equations of linear elasticity and solve a sample problem. The XML code representing the finite element mesh is left out. The file is part of the tutorial example `A-linear/08-system` in the repository `hpfem/hermes-tutorial` on Github[1]. All other steps of the algorithm are explained via comments in the code below:

[1] https://github.com/hpfem/hermes-tutorial.git.

```
# Initial polynomial degree.
P_INIT = 6

# Problem parameters.
E = 200e9
nu = 0.3
rho = 8000.0
g1 = -9.81
f0 = 0.0
f1 = 8e4

# Import Hermes.
import hermes_common
import hermes2d

# Create custom elasticity forms.
class PyCustomWeakFormElasticity(hermes2d.PyCustomWeakFormReal):
    def __init__(self, E, nu, rho_g, surface_force_bdy, f0, f1):
        # Specify number of equations.
self.super(2)
        # Lame constants.
lambda = (E * nu) / ((1 + nu) * (1 - 2*nu))
mu = E / (2*(1 + nu))
        # Add weak forms for the Jacobian matrix. Here (0, 0) means
        # the upper left block, (0, 1) the upper right corner, etc.
self.add_matrix_form(hermes2d.PyDefaultJacobianElasticity_0_0(0, 0, lambda, mu))
self.add_matrix_form(hermes2d.PyDefaultJacobianElasticity_0_1(0, 1, lambda, mu))
self.add_matrix_form(hermes2d.PyDefaultJacobianElasticity_1_0(1, 0, lambda, mu))
self.add_matrix_form(hermes2d.PyDefaultJacobianElasticity_1_1(1, 1, lambda, mu))

        # Add default vector forms for the residual. These correspond to the
        # four blocks in the Jacobian matrix.
self.add_vector_form(hermes2d.PyDefaultResidualElasticity_0_0(0, lambda, mu))
self.add_vector_form(hermes2d.PyDefaultResidualElasticity_0_1(0, lambda, mu))
self.add_vector_form(hermes2d.PyDefaultResidualElasticity_1_0(1, lambda, mu))
self.add_vector_form(hermes2d.PyDefaultResidualElasticity_1_1(1, lambda, mu))

        # Add remaining volumetric and surface vector forms. Here '0' means the
        # first equation, '1' the second.
self.add_vector_form_surf(hermes2d.PyDefaultVectorFormSurf(0, -f0,
                                  surface_force_bdy))
self.add_vector_form(hermes2d.PyDefaultVectorFormVol(1, -rho_g))
self.add_vector_form_surf(hermes2d.PyDefaultVectorFormSurf(1, -f1,
                                  surface_force_bdy))

# Initialize mesh.
mesh = hermes2d.PyMesh()

# Initialize mesh reader.
reader = hermes2d.PyMeshReaderH2DXML()

# Load the mesh.
reader.load_stream('SEE TUTORIAL FOR XML CODE', mesh)
```

```
# One level of uniform mesh refinements.
mesh.refine_all_elements()

# Zero displacement, to be applied to both displacement
# components.
bc = hermes2d.PyDefaultEssentialBCConstReal("Bottom", 0.0)

# Container to pass BCs into Space.
bcs = hermes2d.PyEssentialBCsReal(bc)

# Spaces for horizontal and vertical displacement
# components.
u1_space = hermes2d.PyH1SpaceReal(mesh, bcs, P_INIT)
u2_space = hermes2d.PyH1SpaceReal(mesh, bcs, P_INIT)
ndof = u1_space.get_num_dofs() + u2_space.get_num_dofs()
print "Number of DOF:", ndof

# Initialize weak formulation.
# Here "Top" is the surface force boundary.
wf = PyCustomWeakFormElasticity(E, nu, rho*g1, "Top", f0, f1)

# Discrete problem.
dp = hermes2d.PyDiscreteProblemReal(wf, [u1_space, u2_space])

# Initialize Newton solver.
newton = hermes2d.PyNewtonSolverReal(dp)

# Create zero coefficient vector.
coef = []
for i in range(ndof): coef.append(0)

# Solve the problem using the Newton's method.
newton.solve(coef)

# Initialize solutions.
u1_solution = hermes2d.PySolutionReal()
u2_solution = hermes2d.PySolutionReal()

# Translate resulting coefficient into Solutions.
hermes2d.PySolutionReal().vector_to_solution(newton.get_sln_vector(), \
    [u1_space, u2_space], [u1_solution, u2_solution])

# Initialize Linearizer.
lin = hermes2d.PyLinearizer()

# Linearize solution and save on server.
lab.writeVTK('~/vtk', lin.save_solution_vtk_to_stream(u1_solution, 'sln'))

# Show solution in Postprocessor.
lab.postprocessor(dict (solution = lab.readVTK('~/vtk')))
```

The output of the script is shown in Fig. 15.

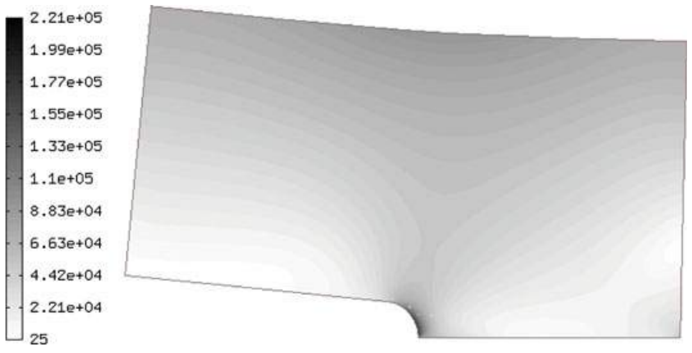


Fig. 15 Output of the linear elasticity example—von Mises stress

References

1. Cloud computing: Wikipedia page http://en.wikipedia.org/wiki/Cloud_computing. Retrieved on September 15, 2012
2. Mongo Database: http://www.mongodb.org/. Retrieved on September 15, 2012
3. Linode: http://www.linode.com/. Retrieved on September 15, 2012
4. Solin P et al (2012) Introduction to Python programming. In: Open source textbook. http://femhub.com/textbook-python. Retrieved on September 15, 2012
5. Scipy official website: http://www.scipy.org/. Retrieved on September 15, 2012
6. Numpy official website: http://numpy.scipy.org/. Retrieved on September 15, 2012
7. Pylab official website: http://www.scipy.org/PyLab. Retrieved on September 15, 2012
8. Matplotlib official website: http://matplotlib.org/. Retrieved on September 15, 2012
9. Sympy official website: http://www.sympy.org/. Retrieved on September 15, 2012
10. GNU Octave http://www.gnu.org/software/octave/. Retrieved on September 15, 2012
11. PyCUDA documentation page: http://documen.tician.de/pycuda/. Retrieved on September 15, 2012
12. NetworkX official website: http://networkx.lanl.gov/. Retrieved on September 15, 2012
13. Hagberg AA, Schult DA, Swart PJ (2008) Exploring network structure, dynamics, and function using NetworkX. In: Proceedings of the 7th Python in Science Conference (SciPy 2008), pp 1115. Pasadena, CA, USA, Aug 2008
14. PLaSM: Wikipedia page http://en.wikipedia.org/wiki/PLaSM. Retrieved on September 15, 2012
15. Paoluzzi A (2003) Geometric programming for computer aided design. Wiley, New York. ISBN 0471899429
16. Solin P et al (2012) Solid modeling with PLaSM. In: Open source textbook. http://femhub.com/textbook-cad. Retrieved on September 15, 2012
17. Hermes: Higher-order modular finite element system. http://hpfem.org/hermes. Retrieved on September 15, 2012
18. Solin P, Segeth K, Dolezel I (2003) Higher-Order Finite Element Methods, Chapman & Hall/CRC Press, New York
19. Shewchuk JR (1996) Triangle: Engineering a 2D quality mesh generator and Delaunay Triangulator. In: Lin MC, Manocha D (eds) Applied computational geometry: towards geometric engineering. Lecture notes in computer science, vol 1148, pp 203–222. Springer, Berlin
20. Python programming language official website: http://www.python.org/. Retrieved on September 15, 2012