Getting started with FEniCSx

Bibekananda Datta
Department of Mechanical Engineering, Johns Hopkins University, Baltimore, MD

March 2023

What is FEniCSx

FEniCSx is an open-source multi-platform computing environment to solve partial differential equations using the finite element method. FEniCSx supports parallel computing with Python and C++ interface. FEniCSx is comprised of the libraries UFL, Basix, FFCx, and DOLFINx which are the build blocks of it. To learn more about FEniCSx: https://fenicsproject.org.

The FEniCS project originally started in 2003 and was known as FEniCS. In 2020, the developers released a new version of the library and renamed FEniCS as FEniCSx. The latest stable version of legacy FEniCS was released in April 2019 and it's barely updated. But many tutorials and legacy codes are perhaps written in legacy FEniCS. So you may want to install a version of it. We will demonstrate installing both versions on multiple different platforms here.

Both FEniCSx and FEniCS are available on Linux, macOS, and Windows. You can download and install it in different ways. Check out the options here for FEniCSx: https://github.com/FEniCS/dolfinx and here for legacy FEniCS: https://fenicsproject.org/download/archive/. For both versions, our preferred approach is to install them via Anaconda.

Installation on Windows

This step is only applicable if you are using Windows. You will have to set up Windows Subsystem for Linux (WSL) to use Anaconda-based installation of FEniCSx and legacy FEniCS. WSL is a virtual Linux environment within Windows that allows you to use Linux command line tools and GUI applications (GUI is natively supported on Windows 11). If you are on macOS or Linux, you can skip this step. If you are on Windows and already have installed WSL, it still might be interesting to download the recommended applications/ tools, such as VS Code.

Get your tools ready first

PowerShell 7

Microsoft Windows already comes with Windows PowerShell 5.1, but the modern edition of PowerShell 7 is more powerful and available on different operating systems.

To learn more about this, check this: https://learn.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-windows.

1. Install the App Installer from the Microsoft App Store. That will enable the `winget` tool on default PowerShell 5.1 and new PowerShell 7.X.Y to be installed.

2. Open the default Windows PowerShell 5.1 as **administrator** from the Windows Start menu, then do the following:

```
$ winget search Microsoft.PowerShell
```

   This command will return the available PowerShell versions to install. We do not recommend installing the `.preview` version.

3. Now install the stable release of PowerShell using the following command:

```
$ winget install --id Microsoft.Powershell --source winget
```

Windows Terminal

Windows Terminal is a command line tool, available on **Microsoft Store** (https://apps.microsoft.com/store/apps). You can manage multiple command line environments such as Windows PowerShell 5.1, PowerShell 7.X.Y, CMD, WSL, etc. using a single application.

Check out more about it here: https://learn.microsoft.com/en-us/windows/terminal/install. You can customize your command line applications using Terminal.

Windows Subsystem for Linux (WSL)

1. The current stable version for Windows Subsystem for Linux is WSL2 and this is the default installation. Open PowerShell 7.X.Y (your current installation version) as **administrator** using the Terminal app from the Windows Start menu and then type the following command:

   ```
   $ wsl --list --online
   ```

   This command will show you currently available Linux distributions on Windows.

2. Install the latest LTS version of Ubuntu which is Ubuntu 22.04.02 LTS.

   ```
   $ wsl --install -d Ubuntu
   ```

   It will ask you to create a user account and set a password. The installation process is fast and straightforward.

3. Once Ubuntu in WSL is installed, you can use it similarly to a regular Ubuntu distribution. To run Ubuntu, open it from the Terminal application option. If this is your first time using Linux, you can familiarize yourself with some commands and operations from here: https://ubuntu.com/tutorials/command-line-for-beginners.

4. We will now update the Ubuntu distribution and install two packages for WSL virtual display settings.

   ```
   $ sudo apt update && sudo apt upgrade
   $ sudo apt install xvfb libgl1-mesa-glx
   ```

   The first command will ask for your password. Proceed as needed.

5. To open Windows like File Explorer from Ubuntu, type:

   ```
   $ explorer.exe .
   ```

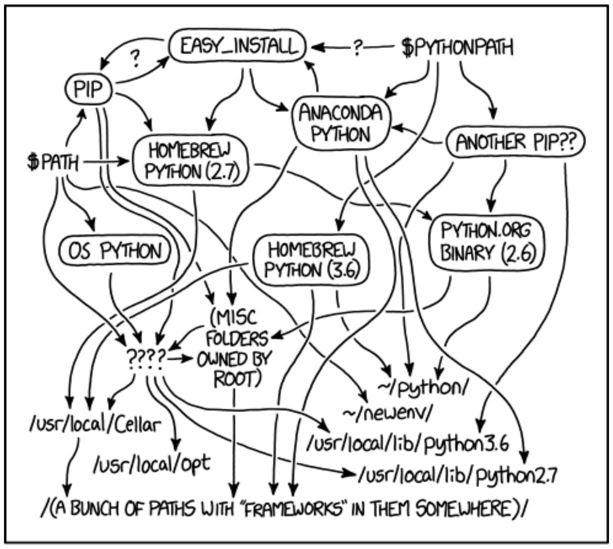
   You can open the WSL home directory and browse, copy, and move files like Windows.

6. Once you have installed Ubuntu in WSL and VS Code, you can follow these tutorials to get yourself more familiarized with the VS Code environments and capabilities in WSL: `https://learn.microsoft.com/en-us/windows/wsl/tutorials/wsl-vscode` and `https://code.visualstudio.com/docs/remote/wsl-tutorial`.

7. To use packages and libraries installed in WSL, start VS Code from the Ubuntu terminal.

Anaconda on Linux (WSL) and macOS

If this is the first time you are using Python, then sit back, it is going to get confusing like this xkcd comic. Even if you are experienced in Python, you might find this interesting.



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Most of the Python-based libraries and packages often depend on other libraries and packages (called dependencies). Installing the right version of those packages and maintaining them is often a complex task even for experienced developers. So, we will use a popular Python distribution, Anaconda, to manage all the Python-based libraries and relevant dependencies. Anaconda uses the conda package manager and works well with pip (Python's default package manager). Installation of Anaconda comes with popular Python packages such as NumPy, SciPy, Matplotlib, and <*insert hundred other packages here*>. It also comes with Jupyter Notebook and Spyder IDE. Anaconda also has a minimalist version, called Miniconda, which is lightweight because it only includes the conda manager and a few packages. We strongly recommend installing Anaconda.

ANACONDA

1. This step is only for macOS. We will install `xcode` command line tools to enable basic development libraries and features. On the macOS terminal, type:

   ```
   (base) $ xcode-select --install
   ```

2. Download Anaconda from here based on your operating system and architecture: https://www.anaconda.com/products/distribution.

3. On macOS, you can download the graphical installer and install it like every other software by following the on-screen instructions. Details of the installation procedure are available here: https://docs.anaconda.com/anaconda/install/mac-os/. Alternatively, you can download the installation script and use the command line to install Anaconda like Linux.

4. For WSL, you will download the installation script. Open the Ubuntu terminal and use the following command to copy this script from your Windows Downloads folder to WSL home directory:

   ```
   $ wget https://repo.anaconda.com/archive/Anaconda3-2023.03-Linux-x86_64.sh
   ```

   `Anaconda3-2023.03-Linux-x86_644` is the latest version of Anaconda available at the time this guide is being written. Check the available versions here: https://repo.anaconda.com/archive. Replace it with the version you downloaded for installation.

5. Once it is copied into the home directory, you can run the following command on the Ubuntu terminal for installation:

   ```
   $ bash Anaconda3-2022.10-Linux-x86_64.sh
   ```

   Please make sure to use the version you downloaded. It will ask you to read the license agreement and your permission for installation. Proceed as needed.

6. Close the Ubuntu terminal and open it back, you will see the base environment for Anaconda is now available. This is how your terminal is going to look like:

   ```
   (base) $
   ```

7. Check out these operations and commands for using Conda: https://docs.conda.io/projects/conda/en/4.6.0/_downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf

FEniCSx on Linux and macOS

As of March 2023, the latest stable release of dolfinx available via Anaconda is 0.6. If you build FEniCSx from the source, you can perhaps install 0.7.

1. Once Anaconda is properly installed, create an environment for FEniCSx. In addition to the dolfinx library, we will install mpich, pyvista, matplotlib, and cycler,. mpich allows parallel processing of different operations within FEniCSx and the other three packages are used for quick visualization. Standard installation of Anaconda already comes with these three packages but we will still need to install them inside the FEniCSx environment.

```
(base)    $ conda create -n fenicsx
(base)    $ conda activate fenicsx
(fenicsx) $ conda install -c conda-forge fenics-dolfinx mpich pyvista
   matplotlib cycler
```

   pyvista supports plotting higher-order unstructured mesh. matplotlib lacks support for higher-order unstructured mesh. So, it is recommended to use pyvista for quick visualization. But you can use matplotlib for regular plotting.

2. To uninstall FEniCSx packages from Anaconda (Only do this step whenever you need to uninstall/ re-install the package), you will have to uninstall everything within the environment. Before you proceed to uninstall check if FEniCSx environment is active in the terminal. If it is active, then deactivate it first and proceed to uninstall the packages.

```
(fenicsx) $ conda deactivate
(base)    $ conda remove -n fenicsx --all
(base)    $ conda clean --all
```

   It will ask your permission; proceed as needed. FEniCSx should be completely uninstalled now.

Legacy FEniCS on Linux and macOS (optional)

A lot of the tutorials, examples, and published codes are still written in legacy FEniCS. So, you may want to install the legacy version in case you want to run codes written in legacy FEniCS.

1. Installation process is similar to FEniCSx. Since higher order mesh wasn't a feature for legacy FEniCS, we are skipping the installation of pyvista here. Necessary visualization can be done using matplotlib.

```
(base)   $ conda create -n fenics
(base)   $ conda activate fenics
(fenics) $ conda install -c conda-forge fenics matplotlib cycler
```

2. Uninstallations procedure for FEniCS (only follow this step if you want to uninstall, skip otherwise) from Anaconda is similar to the FEniCSx package.

```
(fenics) $ conda deactivate
(base)   $ conda remove -n fenics --all
(base)   $ conda clean --all
```

An alternative way to install on Ubuntu (not recommended)

1. On Ubuntu, we can also install FEniCSx using `apt` package manager. Albeit the installation process is simple and lightweight, FEniCSx version available via `apt` is often not the latest version. So, we do not recommend installing this way.

```
$ sudo add-apt-repository ppa:fenics-packages/fenics
$ sudo apt update
$ sudo apt install fenicsx
```

2. To uninstall (Only do this step whenever you need to uninstall/ re-install the package) FEniCSx using `apt` on Ubuntu, follow the procedures below:

```
$ sudo apt remove fenicsx
$ sudo apt remove --auto-remove fenicsx
$ sudo apt purge fenicsx
$ sudo apt purge --auto-remove fenicsx
```

3. Legacy FEniCS can be installed using this approach as well. Please make sure to use `fenics` instead of `fenicsx` in the command line for installing and uninstalling legacy FEniCS.

Visual Studio Code (optional but recommended)

Visual Studio Code is a cross-platform code editor from Microsoft. You can install different extensions within VS Code to enable features for code development such as remote SSH, debugging, compiling, etc. If you have another favorite IDE (e.g., Spyder or PyCharm) or code editor (e.g., Sublime Text, Atom, Notepad++) you like to use, you can configure it yourself.

1. Download VS Code from here: https://code.visualstudio.com/download for your operating system. Follow graphical instructions for installation.
2. If you want to open an empty file in your current working directory using VS Code, then type:

```
$ code .
```

3. To open a file using VS Code from the terminal, go to the directory from the terminal and type:

```
$ code filename.ext
```

4. Once installed, add the extension packs for Python, Remote Development, and Jupyter, from Microsoft. These packages will be necessary for smooth operation. Do a Google search to find out a few more extension packs of your liking. You can also install `Vim` extension pack as well but exit at your own risk.

When you finally exit vim
You know, I'm something of a scientist myself

Test your FEniCSx installation

1. We will now demonstrate how to run a simple FEniCSx example code here using VS Code. If you open the Ubuntu or macOS terminal now, you will see the `(base)` environment is

active. Activate the `(fenicsx)` environment before running the code.

```
(base) $ conda activate fenicsx
```

Now we should see:

```
(fenicsx) $
```

In case you close your terminal and reopen it, you will see the `(base)` environment is active by default. You will have to activate `(fenicsx)` environment using the above command.

2. Create a directory called `fenicsx-code` in your WSL home directory and navigate to it:

```
(fenicsx) $ mkdir fenicsx-code
(fenicsx) $ cd fenicsx-code
```

For macOS, you can follow the same procedure to create a directory. For any operating system, you can create a directory somewhere else as well.

3. Open a new file using VS Code from the Ubuntu terminal:

```
(fenicsx) $ code .
```

4. Copy the following Python code and save the file as `poisson.py`. This code solves a simple 2D Poisson problem. Technical details of this code are described here: https://jsdokken.com/dolfinx-tutorial/chapter1/fundamentals.

```python
import os
import numpy
import ufl
import dolfinx
from dolfinx import mesh, fem, io, plot
from mpi4py import MPI
from petsc4py import PETSc
import pyvista as pv

# clears the terminal and prints dolfinx version
os.system('clear')
# prints dolfinx version
print(f"DOLFINx version: {dolfinx.__version__}")
```

```python
# no of elements in each direction
NElem   = 8

# create a unit square with 8x8 elements with quad elements and use
↪  first order shape function
domain  = mesh.create_unit_square(MPI.COMM_⌋
↪  WORLD,NElem,NElem,mesh.CellType.quadrilateral)
V       = fem.FunctionSpace(domain,("CG",1))

## define trial and test functions
u       = ufl.TrialFunction(V)
v       = ufl.TestFunction(V)

# source term of the poisson equation
f       = fem.Constant(domain, PETSc.ScalarType(-6))

## applying boundary conditions
uD      = fem.Function(V)
uD.interpolate(lambda x: 1 + x[0]**2 + 2 * x[1]**2)
tdim    = domain.topology.dim
fdim    = tdim - 1
domain.topology.create_connectivity(fdim, tdim)
boundary_facets = mesh.exterior_facet_indices(domain.topology)
boundary_dofs   = fem.locate_dofs_topological(V, fdim, boundary_facets)
bc      = fem.dirichletbc(uD, boundary_dofs)

# bilinear form
a       = ufl.dot(ufl.grad(u), ufl.grad(v)) * ufl.dx
L       = f * v * ufl.dx

# set PETSc solver options
sol_opts = {"ksp_type": "preonly", "pc_type": "lu"}
# formulate the problem
problem = fem.petsc.LinearProblem(a, L, bcs=[bc], petsc_options=sol_opts)
# solve the problem
uh = problem.solve()

## error calculation
V2          = fem.FunctionSpace(domain, ("CG", 2))
```

```
uex         = fem.Function(V2)
uex.interpolate(lambda x: 1 + x[0]**2 + 2 * x[1]**2)

L2_error    = fem.form(ufl.inner(uh - uex, uh - uex) * ufl.dx)
error_local = fem.assemble_scalar(L2_error)
error_L2    = numpy.sqrt(domain.comm.allreduce(error_local, op=MPI.SUM))

error_max   = numpy.max(numpy.abs(uD.x.array-uh.x.array))

# print the error
if domain.comm.rank == 0:
    print(f"Error_L2 : {error_L2:.2e}")
    print(f"Error_max : {error_max:.2e}")

# writing output files in xdmf format
with io.XDMFFile(domain.comm, "output.xdmf", "w") as xdmf:
    xdmf.write_mesh(domain)
    xdmf.write_function(uh)

# 2D contour plots of the mesh and result using pyvista
#pv.start_xvfb()    # Uncomment the line on WSL
pv.off_screen = True
topology, cell_types, geometry = plot.create_vtk_mesh(domain, tdim)
grid = pv.UnstructuredGrid(topology, cell_types, geometry)

plotter = pv.Plotter()
plotter.add_mesh(grid,show_edges=True)
plotter.view_xy()
plotter.save_graphic('mesh.pdf')

u_topology, u_cell_types, u_geometry = plot.create_vtk_mesh(V)
u_grid = pv.UnstructuredGrid(u_topology, u_cell_types, u_geometry)
u_grid.point_data["u"] = uh.x.array.real
u_grid.set_active_scalars("u")
u_plotter = pv.Plotter()
u_plotter.add_mesh(u_grid,show_edges=True)
u_plotter.view_xy()
u_plotter.save_graphic('contour.pdf')
```

5. Now run the Python code from the Ubuntu terminal:

```
(fenicsx) $ python3 poisson.py
```

6. This should save `.h5` and `.xdmf` files with the results and save the mesh and contour plot of the primary variable `.pdf` files in the working directory.

A few more settings

If this is the first time you are setting up Python and Anaconda on macOS or WSL, you may need to follow a few more steps for smooth operation. If you have used the Anaconda + VS Code combination before, you might find some of these steps redundant.

1. We will now install a few more Python packages to run FEniCSx code interactively from VS Code.

```
(fenicsx) $ conda install –c anaconda ipykernel ipywidgets
(fenicsx) $ pip install trame
```

   `trame` and `ipywidgets` are needed for Jupyter backend rendering within VS Code Jupyter notebook. For interactive running, we will need to install `ipykernel`.

2. To use mesh generating software, Gmsh, Python interface library for Gmsh, PyGmsh, and mesh converted library, meshio, install the following:

```
(fenicsx) $ pip install ––upgrade gmsh
(fenicsx) $ pip install pygmsh
(fenicsx) $ pip install meshio
```

   Gmsh is an open-source mesh generation tool, available on Linux, macOS, and Windows. Gmsh has API for C++, Python, Julia, etc. PyGmsh is a package built on top of Gmsh's Python API to make mesh generation more convenient. Meshio is a package that can convert meshing files (to and from) into different formats.

3. We will primarily use VS Code + Python for FEniCSx. If you use VS Code + Python for other tasks, you may not want to set `fenicsx` environment as your default environment. We will now get the directory for our `fenicsx` environment within Anaconda:

```
(fenicsx) $ conda info ––envs
```

   This will return all the environments within Anaconda. Copy the directory for `fenicsx` environment.

4. Open VS Code from macOS or Ubuntu terminal using:

   ```
   (fenicsx) $ code .
   ```

   On macOS, go to **Code** > **Settings** > **Settings** > **Extensions** > **Python**. Scroll down to **Default Interpreter Path** and enter `/Users/username/opt/anaconda3/envs/fenicsx/bin/python` (copied directory for `fenicsx`) as the **Default Interpreter Path**.

   On Windows or Linux, go to **File** > **Preferences** > **Settings** > **Extensions** > **Python**. Scroll down to **Default Interpreter Path**, click on **Also modified in Remote**, and enter `/home/username/anaconda3/envs/fenicsx/bin/python` (copied directory for `fenicsx`) as the **Default Interpreter Path**.

5. For both macOS and WSL, scroll down further and check on the boxes for **Terminal: Activate Environment** and **Terminal: Execute In File Dir** options. These will ensure your Python environment is active and you are launching Python in the working directory when running codes from VS Code.

6. This step is only for WSL. If you want to save the PyVista plots in `.png` format with `plotter.screenshot()` function on WSL, you have to configure the remote display option. Open your `bash` profile:

   ```
   (fenicsx) $ code ~/.bashrc
   ```

   Add the following block in your `.bashrc` file:

   ```
   export DISPLAY=:99.0
   export PYVISTA_OFF_SCREEN=true
   Xvfb :99 –screen 0 1024x768x24 > /dev/null 2>&1 &
   sleep 1
   ```

   Save the file and close it. Close and restart your Ubuntu terminal.

7. When you re-open the Ubuntu or macOS terminal, you will notice `(fenicsx)` environment is no longer active. If you want this environment to be active at the terminal start-up then open the `.bashrc` or `.zshrc` file using VS Code and add the following line:

   ```
   conda activate fenicsx
   ```

   Save the file and close it. Restart your terminal. Now you should see `(fenicsx)` environment is always active. If you want to return to the `(base)` conda environment, then type:

```
(fenicsx) $ conda deactivate
```

This will return the following on your terminal:

```
(base) $
```

Known issues with PyVista

1. PyVista plot window needs to be closed before proceeding to the next action (saving the file or plotting another image). To take a screenshot on Windows (or WSL), use `q` to exit the plot window instead of clicking on `X` icon.

2. WSL on Windows 10 lacks native X-11 forwarding support. When the code is run from the terminal or a Python file from VS Code, plotting through `PyVista` will not show the plot unless X-11 server is configured somehow. It can save the plot as `.pdf, .eps, .svg, .tex, .ps` file using `save_graphic()` function or as `.png` file as `plotter.screenshot()` function. This might not be an issue on Windows 11 as WSL natively supports X-11 forwarding (not tested yet).

3. If you are running the Python code interactively from VS Code on WSL, it can show the interactive plot within the Jupyter notebook environment but it has trouble saving the screenshot as `.png` file with `show_edges=True` option. Best is to use the `save_graphic()` function and save it `.pdf, .eps, .svg, .tex, .ps` file.

4. PyVista has trouble rendering higher order ($p \geq 2$) Quadrilateral and Hexahedron elements. For the 2nd order Hexahedron element, it renders Tetrahedron. Visualization in ParaView works properly.