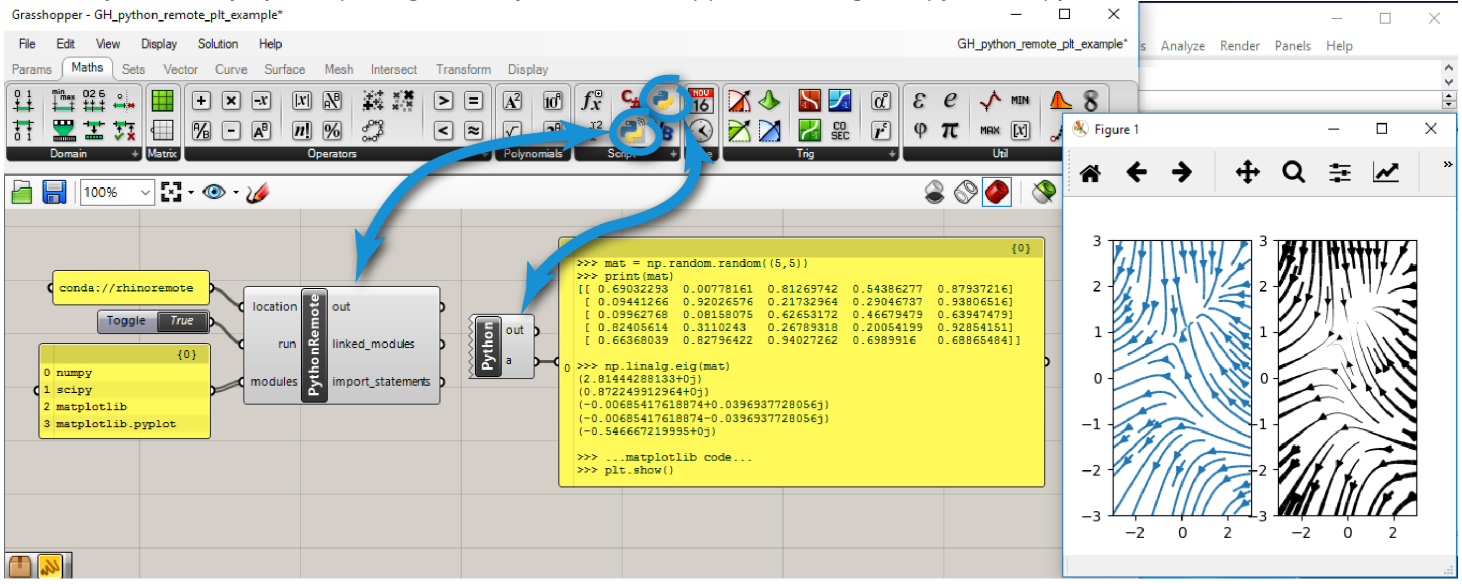
gh-python-remote

Connect an external python instance to Grasshopper, and vice-versa.
This lets you run any Python package directly from Grasshopper, including numpy and scipy!



```
>>> mat = np.random.random((5,5))
>>> print(mat)
[[ 0.69032293  0.00778161  0.81269742  0.54386277  0.87937216]
 [ 0.09441266  0.92026576  0.21732964  0.29046737  0.93806516]
 [ 0.09962768  0.08158075  0.62653172  0.46679479  0.63947479]
 [ 0.82405614  0.3110243   0.26789318  0.20054199  0.92854151]
 [ 0.66368039  0.82796422  0.94027262  0.6989916   0.68865484]]

>>> np.linalg.eig(mat)
(2.81444288133+0j)
(0.872249912964+0j)
(-0.00685417618874+0.0396937728056j)
(-0.00685417618874-0.0396937728056j)
(-0.546667219995+0j)

>>> ...matplotlib code...
>>> plt.show()
```

Installation

Requires a Python 2.7 installation, not compatible with Python 3. Python 2.6 might work but is not supported.

Uses rpyc for the connection backend (automatically installed).

1. Install the software dependencies:

Python 2.7 (2.6 might work too):	gh-python-remote was developed with the Anaconda distribution in mind (comes with numpy and scipy included), but anything else works. If you already have Anaconda installed with Python 3, do not reinstall, instead just read the next paragraph.
Python virtual environment(optional):	isolate dependencies for each project by creating a new virtual environment. If you use Anaconda, `conda env --name rhinoremote --python=2.7` will set you up with a new virtual environment named `rhinoremote`.
Rhinoceros3D:	version 5 is the only one supported by gh-python-remote, no other version works.
Grasshopper:	version 0.9.0076 is supported by gh-python-remote. Version 0.9.0061 and up might work as well. **Open it at least once before continuing.**
GH Python:	version 0.6.0.3 works best, older versions are buggy with gh-python-remote. **Drop it on the Grasshopper canvas at least once before continuing.**

2. Install gh-python-remote:

From the Windows command line (or the special Anaconda or Python command if pip is not in your path by default), run: *(If you are using a virtual environment, remember to* `activate` *it first.)*

```
pip install gh-python-remote --upgrade --process-dependency-links --no-binary=:all:
python -m ghpythonremote._configure_ironpython_installation
```

The first line installs gh-python-remote in the current Python interpreter. The second tries to find your Rhinoceros IronPython installation, and install gh-python-remote there. The extra options are necessary to be able to get the pre-release version of rpyc.

If you do not use the standard Rhinoceros IronPython installation (`%APPDATA%\McNeel\Rhinoceros\5.0\Plug-ins\IronPython (814d908a-e25c-493d-97e9-ee3861957f49)\settings`), you can specify a target directory to use like so: `python -m ghpythonremote._configure_ironpython_installation "location"` .

This will also install the gh-python-remote UserObject in Grasshopper.

Usage

All the examples files are copied in the `%APPDATA%\Grasshopper\UserObjects\gh-python-remote\examples` *folder. You can also download them from the* github repo.

From Grasshopper to Python

1. Open the example file `GH_python_remote.ghx` in Python, or drop the gh-python-remote component on the canvas.
2. Use the `location` input to define the location of the Python interpreter you want to connect to. You can use the path to a folder containing python, the full path to a python executable, or `conda://` followed by the name of an Anaconda virtual environment.
3. Use the `modules` input to define the modules you want to access in the GHPython component. Anything that can follow an `import` statement in the remote Python works. If you need to import a submodule inside a package (like `import this.that`), the parent package has to be imported first.
4. Change `run` to `True` to connect.
5. In the GHPython component, the imported modules will be available via the sticky dictionary. For example if you are trying to use Numpy:

   ```
   import scriptcontext
   np = scriptcontext.sticky['numpy']
   ```

6. Done!

Notes

Creating remote array-like objects from large local lists can be slow. For example, `np.array(range(10000))` takes more than 10 seconds on most computers. To solve this, you need to send the list first to the remote Python interpreter, then create the array from this remote object:

```
import scriptcontext as sc
import rpyc
np = sc.sticky['numpy']
rpy = sc.sticky['rpy']

r_range = rpyc.utils.classic.deliver(rpy, range(10000))
np.array(r_range)
```

There is also an issue that Grasshopper does not recognize remote list objects as lists. They need to be recovered to the local interpreter first:

```
import scriptcontext as sc
import rpyc
from ghpythonlib.treehelpers import list_to_tree # Rhino 6 only!
np = sc.sticky['numpy']

a = np.arange(15).reshape((3,5))
a = rpyc.utils.classic.obtain(a.tolist())
a = list_to_tree(a, source=[0,0])
```

`ghpythonlib.treehelpers` is Rhino 6 only, see the treehelpers gist for an equivalent implementation:

```
def list_to_tree(input, none_and_holes=True, source=[0]):
    """Transforms nestings of lists or tuples to a Grasshopper DataTree"""
    from Grasshopper import DataTree as Tree
    from Grasshopper.Kernel.Data import GH_Path as Path
    from System import Array
    def proc(input,tree,track):
        path = Path(Array[int](track))
        if len(input) == 0 and none_and_holes: tree.EnsurePath(path); return
        for i,item in enumerate(input):
            if hasattr(item, '__iter__'): #if list or tuple
                track.append(i); proc(item,tree,track); track.pop()
            else:
                if none_and_holes: tree.Insert(item,path,i)
                elif item is not None: tree.Add(item,path)
    if input is not None: t=Tree[object]();proc(input,t,source[:]);return t
```

Quick-ref:

** marks an input that is only available by editing the gh-python-remote UserObject, or in* `GH_python_remote.ghx` .

Arguments:	***code (string):**	Path to the `GH_to_python_sticky.py` code file.
	location (string):	Path to a python executable, or to a folder containing `python.exe` , or the name of a conda-created virtual environment prefixed by `conda://` (`conda://env_name`). If empty, finds python from your windows `%PATH%` .
	run (boolean):	Creates the connection, and imports new modules, when turned to True. Kills the connection, and deletes the references to the imports, when turned to False.
	modules (string list):	List of module names to import in the remote python. They will be added to the `scriptcontext.sticky` dictionary, allowing them to be reused from other python components in the same Grasshopper document. Submodules (for example `numpy.linalg` have to be added explicitly to this list to be available later.
	***log_level (string from ['NOTSET', 'DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL']):**	
		Logging level to use for the local IronPython and the remote python instance.
	***working_dir (string):**	Working directory for the remote python instance.
Returns:	**out (string):**	Console output with DEBUG information.
	linked_modules (string list):	list of imported module names.

rpy (rpyc connection object):	The object representing the remote Python interpreter.
import_statements (string):	what to use in the GHPython component to actually use the imported modules.

From Python to Grasshopper

You can also use gh-python-remote to programmatically control a Rhinoceros instance, and connect to it via Python. Have a look at `examples/python_to_GH.py` for a full working example.

License