

Python Native - opis realizacji

20 maja 2020

Spis treści

1	Biblioteka z kodem natywnym	2
1.1	Kod pakietu	2
1.2	Wersja dystrybucyjna	3
2	Wykorzystanie biblioteki/pakietu w aplikacji	3
2.1	Rozwiązywanie pojedynczych równań od użytkownika	4
2.2	Rozwiązywanie równań w danym formacie z pliku	5
2.2.1	Generowanie losowych układów równań	5
2.2.2	Rozwiązywanie równań z pliku	6
3	Wersja wykonywalna	7
4	Sposoby uruchamiania na sklonowanym repozytorium	7
4.1	Wykonanie skryptu i wykorzystanie <i>invoke</i>	7
4.2	Utworzenie pliku wykonywalnego	8
4.3	Uruchomienie pliku wykonywalnego	8

1 Biblioteka z kodem natywnym

Całość projektu została wykonana pod Linuxem, a konkretnie Ubuntu 20.4

1.1 Kod pakietu

Kod biblioteki znajduje się w pliku `.../src/wrapper.cpp` i klasa `Solver` udostępnia jedną główną metodę służącą do rozwiązania układu równań po podaniu parametrów, jakimi są - liczba niewiadomych, współczynniki przy niewiadomych oraz prawe strony równania. Akceptowane są równania typu:

$$\begin{cases} 1 \cdot x_0 + 2 \cdot x_1 = 3 \\ 4 \cdot x_0 + 5 \cdot x_1 = 6 \end{cases}$$

Metoda zapisuje w pamięci albo rozwiązanie równania albo informację, że układ jest sprzeczny. Deklaracja metody wygląda następująco:

```
1 void solve(int n, py::list left, py::list right);
```

gdzie n to liczba niewiadomych (i równań), *left* to lista list współczynników przy x_i czyli dla powyższego przykładu `[[1,2], [4,5]]`, a *right* to lista wartości po prawej stronie równania, czyli dla powyższego przykładu `[3,6]`. Rozwiązania są przechowywane poczynawszy od adresu zawartego we wskaźniku `*X` (będącego polem klasy), ale żeby najpierw je wydobyć sprawdzana jest wartość zmiennej logicznej w polu `solved`, które jest równe `false` jeśli układ okazał się sprzeczny i `true` (domyślnie) jeśli układ został rozwiązany poprawnie. Ale to już obsługuje aplikacja.

Aby kod natywny mógł skorzystać ze zmiennych typu `py::list` została zawarta w nagłówku biblioteka i określony następujący *namespace* (zgodnie z tym co jest zawarte w oficjalnej [dokumentacji pybind11](#)):

```
1 #include <pybind11/pybind11.h>
2 ...
3 namespace py = pybind11;
```

Żeby kod mógł zostać wywołany z poziomu kodu Python'owego zostało zdefiniowane następujące makro:

```
1 PYBIND11_MODULE(equations_solver, m) {
2     m.doc() = "plugin for solving linear equations";
3     py::class_<Solver>(m, "Solver")
4     .def(py::init<>())
5     .def("solve", &Solver::solve, "Method solving systems of linear equations")
6     .def("__repr__",
7         [](const Solver &a) {
8             std::string results = "";
9             for(int i = 0; i < a.size; i++)
10             {
11                 results += std::to_string(a.X[i]) + ":";
12             }
13             if(!a.solved){
14                 results = "false";
15             }
16             return results;
17         }
18     );
19 }
```

Nazwa modułu została umieszczona jako pierwszy argument makro i została określona jako `equations_solver`. Argument *m* definiuje zmienną typu `py::module`, która jest głównym interfejsem do tworzenia bind'ów (powiązań). Metoda `solve` została zbind'owana na metodę o tym samym aliasie (czyli w kodzie Python'a będzie ją można wywołać jako `solve`), a za pomocą metody `__repr__` można dostać się do rozwiązania (w formacie $x_0 : x_1 : \dots : x_n$) -

jeśli takie istnieje, bo jeśli nie (*!a.solved*) to zwracany jest string *false*. Metoda `__repr__` to coś w stylu *toString* w Javie i służy reprezentacji obiektu, a w kodzie Python'a może zostać wywołane poprzez `s = str(p)`, gdzie *p* jest obiektem klasy *Solver* (wtedy w *s* są zapisane rozwiązania układu równań).

1.2 Wersja dystrybucyjna

Utworzenie wersji dystrybucyjnej, czyli de facto biblioteki z rozszerzeniem *.so* zostało wykonane przy pomocy narzędzia *invoke*.

```
1 @invoke.task
2 def clean(c):
3     """ Remove any built objects """
4     for pattern in ["*.o", "*.so"]:
5         c.run("rm -rf {}".format(pattern))
6
7 def compile_python_module(cpp_name, extension_name):
8     invoke.run(
9         "g++ -O3 -Wall -Werror -shared -std=c++11 -fPIC "
10        "'python3 -m pybind11 --includes' "
11        "-I /usr/include/python3.7 -I . "
12        "{0} "
13        "-o {1} 'python3.8-config --extension-suffix' "
14        "-L. -Wl,-rpath,. ".format(cpp_name, extension_name)
15    )
16
17
18 @invoke.task()
19 def build_solver(c):
20     """ Build the pybind11 wrapper library """
21     print_banner("Building PyBind11 Module for solving systems of linear equations")
22     compile_python_module("wrapper.cpp", "equations_solver")
23     print("** Complete")
24
25
26 @invoke.task(
27     build_solver,
28 )
```

W pliku *tasks.py* (koniecznie taka nazwa) zostały zdefiniowane powyższe metody, dzięki czemu zbudowanie biblioteki zawierającej kod natywny, z rozszerzeniem *.so* zostało uproszczone do prostego wywołania komendy:

```
1 invoke build-solver
```

Po wywołaniu tejże komendy na konsoli widać kolejne kroki powodzenia w tworzeniu biblioteki - tak jak na screen-shocie nr 6. Dodatkowo została zdefiniowana metoda *clean* (do wywołania za pomocą *invokeclean*), która „sprząta” w katalogu, czyli usuwa stare pliki z rozszerzeniem *.o* lub *.so*.

Mając już bibliotekę w takiej postaci można ją wykorzystać w Python'ie jako zwykły moduł za pomocą prostego importu, co jest pokazane w następnej sekcji.

2 Wykorzystanie biblioteki/pakietu w aplikacji

Za pomocą zwykłego importu możemy korzystać z metod określonych w zdefiniowanym *makro*:

```
1 import equations_solver
```

Aplikacja wykorzystująca bibliotekę udostępnia użytkownikowi parę metod, ich kod jest zawarty w pliku *.../src/solve.py* i zostanie umieszczony listing tylko tej niewrażliwej metody, która wykorzystuje bibliotekę natywną.

Bibliotekę wykorzystuje się jak zwykły Python'owy moduł:

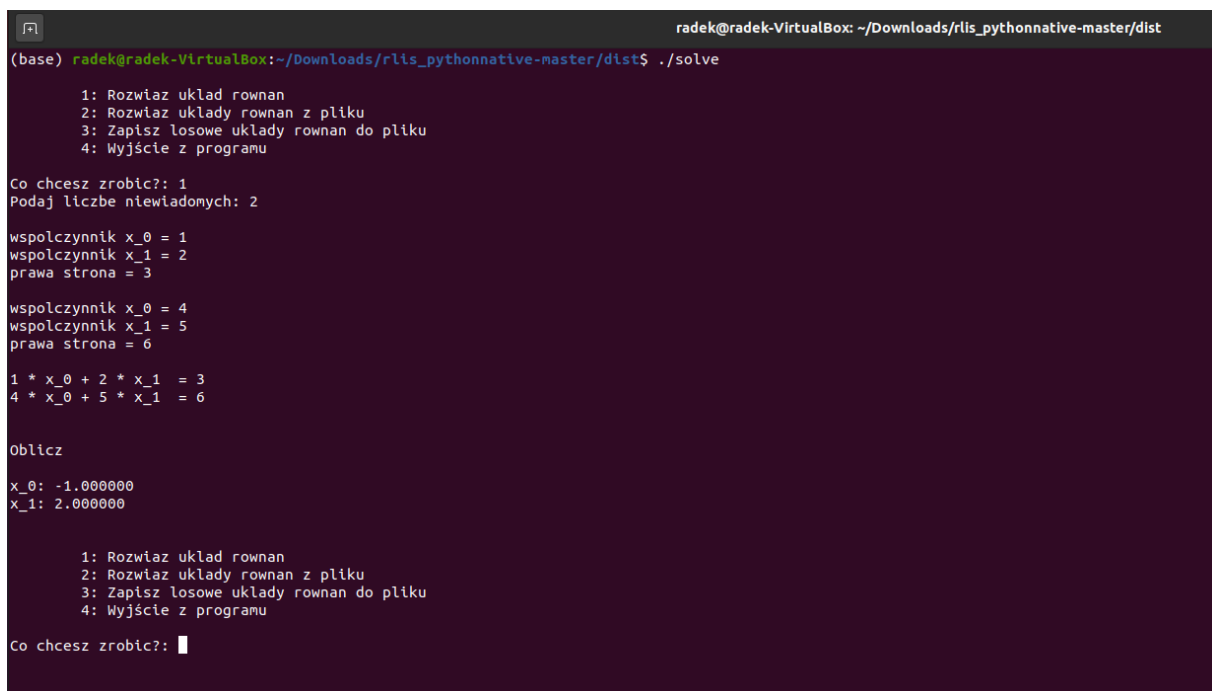
```
1 def solve(size, left_, right_):
2     p = equations_solver.Solver()
3     p.solve(size, left_, right_)
4     values = str(p)[-1].split(':')
5     if len(values) <= 1:
6         print('UKŁAD SPRZECZNY')
7         print()
8         return
9     for index, value in enumerate(values):
10        print(f'x_{index}: {value}')
11    print()
```

Metoda przyjmuje argumenty tak jak zostało to opisane w sekcji 1.1. Tworzy obiekt klasy Solver i następnie wywołuje na nim metodę *solve*, która została zdefiniowana i udostępniona do interfejsu w *makro*. Po wykonaniu metody wywołuje się metodę *__repr__* za pomocą *str(p)*, w której powinny być rozwiązania oddzielone znakiem :, jeśli takich nie ma to zwracana jest informacja, że układ jest sprzeczny. Jeśli są to zostają wyświetlone, co można zobaczyć na następnych screenshot'ach.

2.1 Rozwiązanie pojedynczych równań od użytkownika

Pierwszą z dostępnych metod jest rozwiązywanie równań o współczynnikach pobranych od użytkownika.

Użytkownik podaje liczbę niewiadomych (przyjmowane jest, że jest też taka sama liczba równań) i następnie wpisuje kolejne wymagane współczynniki, a następnie potwierdza, że chce wykonać metodę. Wyświetlane jest rozwiązywane równanie w „w miarę” przyjemnej postaci jak i same rozwiązania:



```
radek@radek-VirtualBox: ~/Downloads/rlls_pythonnative-master/dist
(base) radek@radek-VirtualBox:~/Downloads/rlls_pythonnative-master/dist$ ./solve

1: Rozwiąż układ rownan
2: Rozwiąż układy rownan z pliku
3: Zapisz losowe układy rownan do pliku
4: Wyjście z programu

Co chcesz zrobić?: 1
Podaj liczbę ntewiadomych: 2

wspolczynnik x_0 = 1
wspolczynnik x_1 = 2
prawa strona = 3

wspolczynnik x_0 = 4
wspolczynnik x_1 = 5
prawa strona = 6

1 * x_0 + 2 * x_1 = 3
4 * x_0 + 5 * x_1 = 6

Oblicz

x_0: -1.000000
x_1: 2.000000

1: Rozwiąż układ rownan
2: Rozwiąż układy rownan z pliku
3: Zapisz losowe układy rownan do pliku
4: Wyjście z programu

Co chcesz zrobić?: █
```

Rysunek 1: Rozwiązanie przykładowego równania

2.2 Rozwiązywanie równań w danym formacie z pliku

Możliwe jest wygenerowanie losowych równań (domyślny zakres niewiadomych to 2-3 i wartości współczynników to int'y) w jednolinijkowej postaci do pliku.

2.2.1 Generowanie losowych układów równań

```
radek@radek-VirtualBox: ~/Downloads/rlls_pythonnative-master/dist
(base) radek@radek-VirtualBox:~/Downloads/rlls_pythonnative-master/dist$ ./solve

1: Rozwiąż układ rownan
2: Rozwiąż układy rownan z pliku
3: Zapisz losowe układy rownan do pliku
4: Wyjście z programu

Co chcesz zrobic?: 3
Liczba układów do wygenerowania: 100
Nazwa pliku: data

Poprawnie wygenerowano i zapisano do pliku data

1: Rozwiąż układ rownan
2: Rozwiąż układy rownan z pliku
3: Zapisz losowe układy rownan do pliku
4: Wyjście z programu

Co chcesz zrobic?:
```

Rysunek 2: Wygenerowanie i zapisanie równań do pliku *data*

Równania są postaci:

2:3;7;9;10;9;7

Co oznacza:

$$\begin{cases} 3 \cdot x_0 + 7 \cdot x_1 = 9 \\ 10 \cdot x_0 + 9 \cdot x_1 = 7 \end{cases}$$

```
radek@radek-VirtualBox: ~/Downloads/rlls_pythonnative-master/dist
2:3;7;9;10;9;7
3:5;4;1;3;1;7;4;9;6;4;7;2
3:2;7;9;7;4;1;9;5;6;4;9;3
2:6;9;6;4;5;3
3:4;7;7;6;9;9;8;5;5;6;3;4
2:9;10;9;5;3;1
3:10;8;1;7;4;9;4;9;4;9;1;1
2:9;2;4;10;1;9
3:3;1;4;7;9;2;3;3;9;2;3;5
2:6;8;9;10;3;9
3:7;8;2;5;2;3;3;7;4;4;10;3
3:5;3;7;2;6;10;5;2;10;4;7;2
3:5;9;6;10;6;4;6;5;8;4;10;1
2:3;3;3;1;3;5
3:1;8;7;6;1;3;4;6;10;7;9;9
2:10;6;7;8;9;3
2:6;1;1;10;1;7
3:6;5;1;8;4;7;7;1;3;9;10;1
3:6;4;4;6;9;6;1;7;3;3;8;6
2:1;9;4;1;9;3
3:10;4;7;6;8;2;1;7;6;6;4;2
2:6;1;7;4;6;5
```

Rysunek 3: Zawartość pliku *data*

2.2.2 Rozwiązanie równań z pliku

Aplikacja umożliwia rozwiązanie wygenerowanych równań i prezentację rozwiązań wraz z równaniem:

```
radek@radek-VirtualBox: ~/Downloads/rllis_pythonnative-master/dist

Rownanie nr 95 (2:6;6;2:3;10;9):
x_0: -0.809524
x_1: 1.142857

Rownanie nr 96 (2:10;6;4:9;2;8):
x_0: 1.176471
x_1: -1.294118

Rownanie nr 97 (3:2;7;5;2:6;10;4;2:7;2;3;7):
x_0: 0.750000
x_1: -0.659091
x_2: 1.022727

Rownanie nr 98 (3:10;5;10;1:6;10;9;7:3;3;6;2):
x_0: -0.466667
x_1: 0.854545
x_2: 0.139394

Rownanie nr 99 (3:6;4;3;2:10;9;7;1:5;7;10;9):
x_0: 1.180328
x_1: -3.163934
x_2: 2.524590

Rownanie nr 100 (3:4;8;4;2:6;5;6;7:1;2;1;7):
UKŁAD SPRZECZNY

1: Rozwiąż układ równań
2: Rozwiąż układy równań z pliku
3: Zapisz losowe układy równań do pliku
4: Wyjście z programu
```

Rysunek 4: Rozwiązanie równań z pliku *data*

```
radek@radek-VirtualBox: ~/Downloads/rllis_pythonnative-master/src
radek@radek-VirtualBox:~/Downloads/rllis_pythonnative-master/src$ python3 solve.py

1: Rozwiąż układ równań
2: Rozwiąż układy równań z pliku
3: Zapisz losowe układy równań do pliku
4: Wyjście z programu

Co chcesz zrobić?: 2
Podaj nazwę pliku: dane.txt
=====
ERROR : Niewłaściwie sformatowana 1 linia!
= Niepoprawne wartości - could not convert string to float: 'a.3'
=====
ERROR : Niewłaściwie sformatowana 2 linia!
= Niepoprawny rozmiar - invalid literal for int() with base 10: 'a'
=====
ERROR : Niewłaściwie sformatowana 3 linia!
= Niepoprawny rozmiar - invalid literal for int() with base 10: 'a.1;4.3;5.2'
=====
ERROR : Niewłaściwie sformatowana 4 linia!
= Nie zgadza się liczba wartości
=====

Rownanie nr 5 (2:5.1;4.3;5.2:1.3;9.3;20.6):
x_0: -0.961281
x_1: 2.349426

=====
ERROR : Niewłaściwie sformatowana 6 linia!
= Nie zgadza się liczba wewnętrznych wartości
=====

1: Rozwiąż układ równań
2: Rozwiąż układy równań z pliku
3: Zapisz losowe układy równań do pliku
4: Wyjście z programu

Co chcesz zrobić?:
```

Rysunek 5: Obsługa błędów/wyjątków w niepoprawnie sformatowanych plikach

3 Wersja wykonywalna

Wersja wykonywalna została utworzona za pomocą biblioteki *pyinstaller*, która umożliwia szybkie zbudowanie wersji wykonywalnej. Za pomocą jednej komendy został utworzony plik wykonywalny:

```
pyinstaller --clean -F solve.py
```

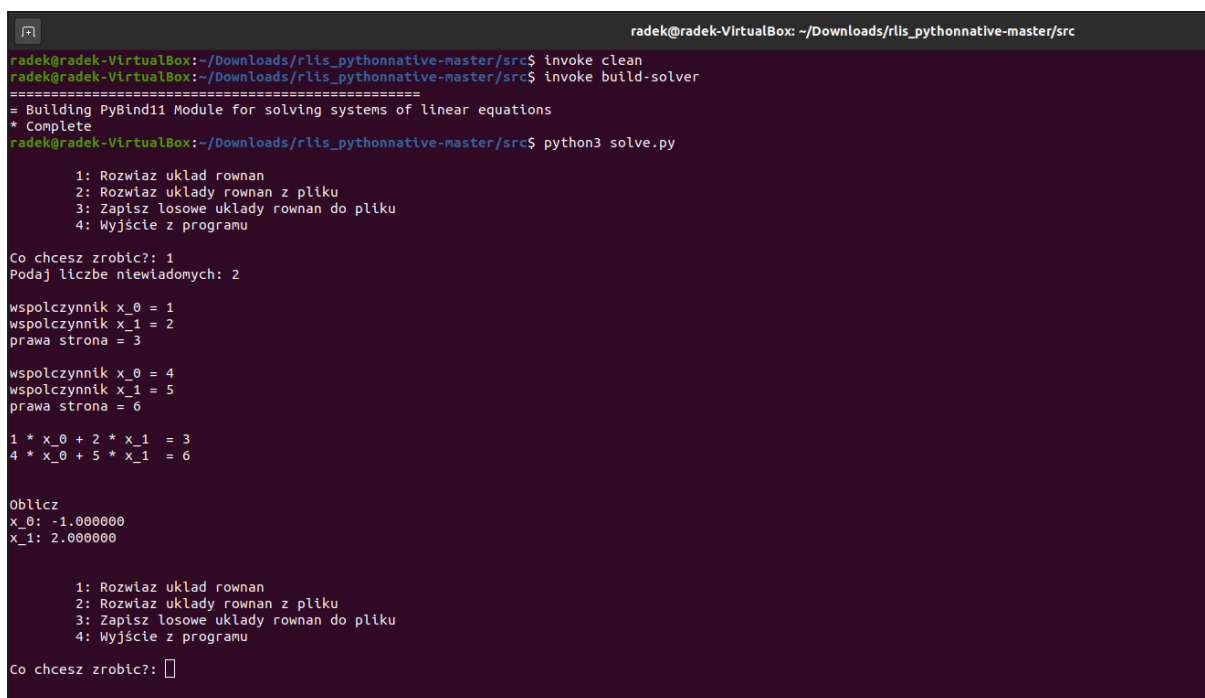
Po wykonaniu komendy tworzy się folder o nazwie *dist*, w którym znajduje się plik gotowy do uruchomienia (*standalone*). W repozytorium można go znaleźć pod ścieżką *.../dist/solve* i uruchomić pod Linux'em. Niestety *pyinstaller* nie jest międzyplatformowy i nie da się uruchomić tego pliku pod Windows'em, ale któraś wersja 1.x *pyinstaller'a* udostępniała taką możliwość.

4 Sposoby uruchamiania na sklonowanym repozytorium

Po sklonowaniu repozytorium można uruchomić aplikację na 3 sposoby:

4.1 Wykonanie skryptu i wykorzystanie *invoke*

Jeśli posiada się Python'a można zbudować bibliotekę za pomocą *invoke* i wykonać skrypt *solve.py* (uprzednio zainstalować zależności (*pip3 install -r requirements.txt*):



```
radek@radek-VirtualBox: ~/Downloads/rlls_pythonnative-master/src
radek@radek-VirtualBox:~/Downloads/rlls_pythonnative-master/src$ invoke clean
radek@radek-VirtualBox:~/Downloads/rlls_pythonnative-master/src$ invoke build-solver
=====
- Building PyBind11 Module for solving systems of linear equations
* Complete
radek@radek-VirtualBox:~/Downloads/rlls_pythonnative-master/src$ python3 solve.py

1: Rozwiaz układ rownan
2: Rozwiaz układy rownan z pliku
3: Zapisz losowe układy rownan do pliku
4: Wyjście z programu

Co chcesz zrobic?: 1
Podaj liczbe niewiadomych: 2

wspolczynnik x_0 = 1
wspolczynnik x_1 = 2
prawa strona = 3

wspolczynnik x_0 = 4
wspolczynnik x_1 = 5
prawa strona = 6

1 * x_0 + 2 * x_1 = 3
4 * x_0 + 5 * x_1 = 6

Oblicz
x_0: -1.000000
x_1: 2.000000

1: Rozwiaz układ rownan
2: Rozwiaz układy rownan z pliku
3: Zapisz losowe układy rownan do pliku
4: Wyjście z programu

Co chcesz zrobic?:
```

Rysunek 6: Utworzenie biblioteki za pomocą *invoke*

4.2 Utworzenie pliku wykonywalnego

Można utworzyć plik wykonywalny i go uruchomić (zainstalować najpierw zależności):

```
radek@radek-VirtualBox: ~/Downloads/rlls_pythonnative-master/src/dist
radek@radek-VirtualBox:~/Downloads/rlls_pythonnative-master/src$ pyinstaller --clean -F solve.py && cd dist
46 INFO: PyInstaller: 3.6
46 INFO: Python: 3.8.2
71 INFO: Platform: Linux-5.4.0-29-generic-x86_64-with-glibc2.29
91 INFO: wrote /home/radek/Downloads/rlls_pythonnative-master/src/solve.spec
93 INFO: UPX is not available.
94 INFO: Removing temporary files and cleaning cache in /home/radek/.cache/pyinstaller
103 INFO: Extending PYTHONPATH with paths
['/home/radek/Downloads/rlls_pythonnative-master/src',
'/home/radek/Downloads/rlls_pythonnative-master/src']
107 INFO: checking Analysis
107 INFO: Building Analysis because Analysis-00.toc is non existent
107 INFO: Initializing module dependency graph...
113 INFO: Caching module graph hooks...
130 INFO: Analyzing base-library.zip ...
4208 INFO: Processing pre-find module path hook distutils
4224 INFO: distutils: retargeting to non-venv dir '/usr/lib/python3.8'
4430 INFO: Caching module dependency graph...
8606 INFO: running Analysis Analysis-00.toc
8666 INFO: Analyzing /home/radek/Downloads/rlls_pythonnative-master/src/solve.py
8689 INFO: Processing module hooks...
9023 INFO: Looking for ctypes DLLs
9099 INFO: Analyzing run-time hooks ...
9184 INFO: Including run-time hook 'pyi_rth_multiprocessing.py'
9152 INFO: Looking for dynamic libraries
9604 INFO: Looking for eggs
9605 INFO: Python library not in binary dependencies. Doing additional searching...
9820 INFO: Using Python library /lib/x86_64-linux-gnu/libpython3.8.so.1.0
9827 INFO: Warnings written to /home/radek/Downloads/rlls_pythonnative-master/src/build/solve/warn-solve.txt
9882 INFO: Graph cross-reference written to /home/radek/Downloads/rlls_pythonnative-master/src/build/solve/xref-solve.html
9902 INFO: checking PYZ
9902 INFO: Building PYZ because PYZ-00.toc is non existent
9902 INFO: Building PYZ (ZlibArchive) /home/radek/Downloads/rlls_pythonnative-master/src/build/solve/PYZ-00.pyz
10512 INFO: Building PYZ (ZlibArchive) /home/radek/Downloads/rlls_pythonnative-master/src/build/solve/PYZ-00.pyz completed successfully.
10521 INFO: checking PKG
10521 INFO: Building PKG because PKG-00.toc is non existent
10521 INFO: Building PKG (CArchive) PKG-00.pkg
13393 INFO: Building PKG (CArchive) PKG-00.pkg completed successfully.
13395 INFO: Bootloader /home/radek/.local/lib/python3.8/site-packages/PyInstaller/bootloader/Linux-64bit/run
13395 INFO: checking EXE
13395 INFO: Building EXE because EXE-00.toc is non existent
13395 INFO: Building EXE from EXE-00.toc
13417 INFO: Appending archive to ELF section in EXE /home/radek/Downloads/rlls_pythonnative-master/src/dist/solve
13480 INFO: Building EXE from EXE-00.toc completed successfully.
radek@radek-VirtualBox:~/Downloads/rlls_pythonnative-master/src/dist$ ./solve

1: Rozwiąż układ rownan
2: Rozwiąż układy rownan z pliku
3: Zapisz losowe układy rownan do pliku
4: Wyjście z programu

Co chcesz zrobić?:
```

Rysunek 7: Zbudowanie wersji wykonywalnej aplikacji

4.3 Uruchomienie pliku wykonywalnego

Można też oczywiście pobrać plik `.../dist/solve` i go uruchomić pod Linux'em, nadając wcześniej odpowiednie uprawnienia:

```
radek@radek-VirtualBox: ~/Downloads/rlls_pythonnative-master/dist
radek@radek-VirtualBox:~/Downloads/rlls_pythonnative-master/dist$ chmod +x solve
radek@radek-VirtualBox:~/Downloads/rlls_pythonnative-master/dist$ ./solve

1: Rozwiąż układ rownan
2: Rozwiąż układy rownan z pliku
3: Zapisz losowe układy rownan do pliku
4: Wyjście z programu

Co chcesz zrobić?:
```

Rysunek 8: Uruchomienie pliku wykonywalnego

Na pewno można ulepszyć sam kod natywny, żeby nie zwracał float'ów tylko ułamki zwykłe.

Bibliografia

- [1] pybind11 - First steps. <https://pybind11.readthedocs.io/en/stable/basics.html>. [Online; accessed 20-May-2020].
- [2] Python Bindings: Calling C or C++ From Python. <https://realpython.com/python-bindings-overview/#pybind11>. [Online; accessed 20-May-2020].