Speeding up Python with Rust

Summer school on modelling and complex systems 2021

Metodi Nikolov

10 July 2021

Contents

Motivating Example

A model

Say that you have come up with a very interesting Bayesian model for the data you have. The model is not supported by the standard packages for inference (e.g. `PyMCMC`, `Stan`), so you will have to code it yourself.

A model

Say that you have come up with a very interesting Bayesian model for the data you have. The model is not supported by the standard packages for inference (e.g. `PyMCMC`, `Stan`), so you will have to code it yourself.

Scaled Student T Distribution

Throughout this we will use a known model, with likelihood function based around the Scaled Student T Distribution, with priors as straightforward as they can be - the idea is to show work flow and ideas, not a specific model.

$$
\begin{aligned}
x &\sim t_\nu(\mu, \sigma^2) \\
\mu &\propto const. \\
\sigma^2 &\propto const.
\end{aligned}
$$

A model

Say that you have come up with a very interesting Bayesian model for the data you have. The model is not supported by the standard packages for inference (e.g. `PyMCMC`, `Stan`), so you will have to code it yourself.

Scaled Student T Distribution

Throughout this we will use a known model, with likelihood function based around the Scaled Student T Distribution, with priors as straightforward as they can be - the idea is to show work flow and ideas, not a specific model.

$$
\begin{aligned}
x &\sim t_\nu(\mu, \sigma^2) \\
\mu &\propto \mathit{const.} \\
\sigma^2 &\propto \mathit{const.}
\end{aligned}
$$

Subordinate Representation

$$x \sim \mu + N\sqrt{\frac{\nu\sigma^2}{\chi^2_\nu}}$$

A model

Another Representation

$$x \sim N(\mu, V)$$
$$V \sim \text{Inv-}\chi^2(\nu, \sigma^2)$$

Source: *Bayesian Data Analysis 3rd ed.* Gelman at al. pp. 295

A model

Another Representation

$$x \sim N(\mu, V)$$
$$V \sim \text{Inv-}\chi^2(\nu, \sigma^2)$$

NB: There is a V for each data point!

Source: *Bayesian Data Analysis 3rd ed.* Gelman at al. pp. 295

A model

Another Representation

$$x \sim N(\mu, V)$$
$$V \sim \text{Inv-}\chi^2(\nu, \sigma^2)$$

NB: There is a V for each data point!

We will cheat a bit (i.e. make the model better in terms of learning it) and extend the above to allow better mixing:

$$x \sim N(\mu, \alpha^2 U)$$
$$U \sim \text{Inv-}\chi^2(\nu, \tau^2)$$

Source: *Bayesian Data Analysis 3rd ed.* Gelman at al. pp. 295

A model

Here, $\alpha^2 U$ plays the role of V, and $\alpha\tau = \sigma - \alpha$ is a 'mixing' parameter, helps with MCMC. The full conditional for Gibbs are as follows:

Source: *Bayesian Data Analysis 3rd ed.* Gelman et al. pp. 295

Here, $\alpha^2 U$ plays the role of V, and $\alpha\tau = \sigma - \alpha$ is a 'mixing' parameter, helps with MCMC. The full conditional for Gibbs are as follows:

Source: *Bayesian Data Analysis 3rd ed.* Gelman et al. pp. 295

A model

Here, $\alpha^2 U$ plays the role of V, and $\alpha\tau = \sigma - \alpha$ is a 'mixing' parameter, helps with MCMC. The full conditional for Gibbs are as follows:

- $$U_i|\alpha, \mu, \tau^2, \nu, \sim \text{Inv-}\chi^2\left(\nu + 1, \frac{\nu\tau^2 + ((x_i - \mu)/\alpha)^2}{\nu + 1}\right)$$

Source: *Bayesian Data Analysis 3rd ed.* Gelman et al. pp. 295

A model

Here, $\alpha^2 U$ plays the role of V, and $\alpha\tau = \sigma - \alpha$ is a 'mixing' parameter, helps with MCMC. The full conditional for Gibbs are as follows:

- $$U_i | \alpha, \mu, \tau^2, \nu, \sim \text{Inv-}\chi^2 \left(\nu + 1, \frac{\nu\tau^2 + ((x_i - \mu)/\alpha)^2}{\nu + 1} \right)$$
- $$\mu | \alpha, \tau^2, U, \nu, x \sim N \left(\frac{\sum \frac{1}{\alpha^2 U_i} x_i}{\sum \frac{1}{\alpha^2 U_i}}, \frac{1}{\sum \frac{1}{\alpha^2 U_i}} \right)$$

Source: *Bayesian Data Analysis 3rd ed.* Gelman et al. pp. 295

A model

Here, $\alpha^2 U$ plays the role of V, and $\alpha\tau = \sigma - \alpha$ is a 'mixing' parameter, helps with MCMC. The full conditional for Gibbs are as follows:

- $$U_i|\alpha, \mu, \tau^2, \nu, \sim \text{Inv-}\chi^2\left(\nu+1, \frac{\nu\tau^2 + ((x_i-\mu)/\alpha)^2}{\nu+1}\right)$$
- $$\mu|\alpha, \tau^2, U, \nu, x \sim N\left(\frac{\sum \frac{1}{\alpha^2 U_i} x_i}{\sum \frac{1}{\alpha^2 U_i}}, \frac{1}{\sum \frac{1}{\alpha^2 U_i}}\right)$$
- $$\tau^2|\alpha, \mu, U, \nu, x \sim \text{Gamma}\left(\frac{\nu n}{2}, \frac{\nu}{2}\sum\frac{1}{U_i}\right)$$

Source: *Bayesian Data Analysis 3rd ed.* Gelman et al. pp. 295

A model

Here, $\alpha^2 U$ plays the role of V, and $\alpha\tau = \sigma - \alpha$ is a 'mixing' parameter, helps with MCMC. The full conditional for Gibbs are as follows:

- $$U_i|\alpha, \mu, \tau^2, \nu, \sim \text{Inv-}\chi^2\left(\nu + 1, \frac{\nu\tau^2 + ((x_i - \mu)/\alpha)^2}{\nu + 1}\right)$$

- $$\mu|\alpha, \tau^2, U, \nu, x \sim N\left(\frac{\sum \frac{1}{\alpha^2 U_i} x_i}{\sum \frac{1}{\alpha^2 U_i}}, \frac{1}{\sum \frac{1}{\alpha^2 U_i}}\right)$$

- $$\tau^2|\alpha, \mu, U, \nu, x \sim \text{Gamma}\left(\frac{\nu n}{2}, \frac{\nu}{2}\sum \frac{1}{U_i}\right)$$

- $$\alpha^2|\mu, \tau^2, U, \nu, x \sim \text{Inv-}\chi^2\left(n, \frac{1}{n}\sum \frac{(x_i - \mu)^2}{U_i}\right)$$

Source: *Bayesian Data Analysis 3rd ed.* Gelman et al. pp. 295

Python implementation

```python
class ScaledTModel(object):

        __slots__ = ['_data', '_data_size', '_nu',...]

        def __init__(self, data, nu):
                pass

        def run(self, burnin, sample_size):
                pass

        def get_mu(self):
                pass

        def get_sigma2(self):
                pass

        def _update_mu(self):
                pass

        def _update_tau2(self):
                pass

        def _update_alpha2(self):
                pass

        def _update_extended_vars(self):
                pass

        def _sampleScaledInvChiSquare(self, ni, scale):
                pass
```

Pure Python implementation

```
def __init__(self, data, nu):
        print("making a model")
        self._data = data
        self._data_size = len(data)
        self._nu = nu

        self._extended_vars = [0.0] * self._data_size
        self._tau2 = 1
        self._mu = sum(data) / self._data_size
        self._alpha2 = 1

        self._update_extended_vars()
```

Pure Python implementation

```
def _update_mu(self):
        variance = 0.0
        expected_value = 0.0
        for i in range(self._data_size):
                tmp = 1.0/self._extended_vars[i]
                variance += tmp
                expected_value += tmp * self._data[i]

        variance /= self._alpha2
        expected_value /= self._alpha2
        variance = 1.0 / variance
        expected_value = expected_value * variance
        self._mu = random.gauss(expected_value, math.sqrt(variance))
```

Pure Python implementation

```
def _update_tau2(self):
        x = 0.0
        for i in range(self._data_size):
                x += 1.0 / self._extended_vars[i]
        self._tau2 = random.gammavariate(
                self._data_size * self._nu / 2.0,
                2.0 / (self._nu * x),
        )

def _update_alpha2(self):
        x = 0.0
        for i in range(self._data_size):
                x += (self._data[i] - self._mu) *
                        (self._data[i] - self._mu) / self._extended_vars[i]
        x /= self._data_size
        self._alpha2 = self._sampleScaledInvChiSquare(self._data_size, x)
```

Pure Python implementation

```
def _update_extended_vars(self):
        for i in range(self._data_size):
                x = (self._data[i] - self._mu) *
                        (self._data[i] - self._mu) / self._alpha2
                self._extended_vars[i] = self._sampleScaledInvChiSquare(
                        self._nu + 1, (self._nu * self._tau2 + x) / (self._nu + 1))

@property
def get_mu(self):
        return self._results_mu

@property
def get_sigma2(self):
        return self._results_sigma2
```

Pure Python implementation

```python
def run(self, burn_in = 1000, sample_size = 2000):
        self._results_mu = [0.0] * sample_size
        self._results_sigma2 = [0.0] * sample_size
        for _ in range(burn_in):
                self._update_extended_vars()
                self._update_alpha2()
                self._update_mu()
                self._update_tau2

        for i in range(sample_size):
                self._update_extended_vars()
                self._update_alpha2()
                self._update_mu()
                self._update_tau2
                self._results_mu[i] = self._mu
                self._results_sigma2[i] = self._alpha2 * self._tau2
```

Pure Python implementation

```python
from great_model.python_great_model import ScaledTModel as PythonModel
import numpy as np
from timeit import timeit

nu = 6
mu = -3.14
sigma2 = 30
data_size = 500
z = np.random.randn(data_size)
x = np.random.chisquare(nu, data_size)

t_data = mu + z * np.sqrt(sigma2 * nu / x)

p_model = PythonModel(t_data, nu)


exec_time = timeit('p_model.run(2000, 2000)',
globals=globals(), number = 10)
print(f"Average run time of Pure Python model: {exec_time * 1000 / 10:3.5f} ms")
Average run time of Pure Python model: 10404.53057 ms
```

That’s a bit more than 10 seconds.

Numpy implementation

```python
def __init__(self, data, nu):
        print("making a model")
        self._data = np.asarray(data)
        self._data_size = len(data)
        self._nu = nu

        self._rng = default_rng()
        # Some starting values
        self._extended_vars = np.zeros(self._data_size)
        self._tau2 = 1
        self._mu = sum(data) / self._data_size
        self._alpha2 = 1

        self._update_extended_vars()

        # temporary data holders, so that we reuse memory
        self._tmp_with_data_size = np.zeros(self._data_size)
        self._tmp_with_data_size2 = np.zeros(self._data_size)
```

Numpy implementation

```python
def _update_mu(self):
        np.reciprocal(self._extended_vars, out=self._tmp_with_data_size)
        self._tmp_with_data_size2 = self._data * self._tmp_with_data_size

        variance = self._tmp_with_data_size.sum()
        expected_value = self._tmp_with_data_size2.sum()

        variance /= self._alpha2
        expected_value /= self._alpha2
        variance = 1.0 / variance
        expected_value = expected_value * variance
        self._mu = self._rng.normal(expected_value, math.sqrt(variance))
```

Numpy implementation

```python
def _update_tau2(self):
        np.reciprocal(self._extended_vars, out=self._tmp_with_data_size)
        x = self._tmp_with_data_size.sum()
        self._tau2 = self._rng.gamma(self._data_size * self._nu / 2.0, 2.0 / (self._nu * x))


def _update_alpha2(self):
        x = 0.0
        self._tmp_with_data_size = self._data - self._mu
        self._tmp_with_data_size = (self._tmp_with_data_size *
                self._tmp_with_data_size) / self._extended_vars
        x = self._tmp_with_data_size.sum()
        x /= self._data_size
        self._alpha2 = self._sampleScaledInvChiSquare(self._data_size, x)
```

Numpy implementation

```python
def _update_extended_vars(self):
        x = (self._data - self._mu) * (self._data - self._mu) / self._alpha2
        self._extended_vars =  self._sampleScaledInvChiSquareN(
                self._nu + 1,
                (self._nu * self._tau2 + x) / (self._nu + 1),
        )

def _sampleScaledInvChiSquareN(self, ni, scale):
        x = self._rng.chisquare(ni, size=len(scale))
        return ni * scale / x
```

Numpy implementation

```
def run(self, burn_in = 1000, sample_size = 2000):
        self._results_mu = np.zeros(sample_size)
        self._results_sigma2 = np.zeros(sample_size)
        for _ in range(burn_in):
                self._update_extended_vars()
                self._update_alpha2()
                self._update_mu()
                self._update_tau2

        for i in range(sample_size):
                self._update_extended_vars()
                self._update_alpha2()
                self._update_mu()
                self._update_tau2
                self._results_mu[i] = self._mu
                self._results_sigma2[i] = self._alpha2 * self._tau2
```

Numpy implementation

```python
from great_model.numpy_great_model import ScaledTModel as NumpyModel
import numpy as np
from timeit import timeit

nu = 6
mu = -3.14
sigma2 = 30
data_size = 500
z = np.random.randn(data_size)
x = np.random.chisquare(nu, data_size)

t_data = mu + z * np.sqrt(sigma2 * nu / x)

n_model = NumpyModel(t_data, nu)


exec_time = timeit('n_model.run(2000, 2000)',
        globals=globals(), number = 10)
print(f"Average run time of Numpy model: {exec_time * 1000 / 10:3.5f} ms")
Average run time of Numpy model: 356.86459 ms
```

That's more like it... Can we do better? And how much better?

Crash Course in Rust

Rust is...

Good because

- A compiled, strongly typed, systems programming language (thing C/C++)
- Fast - about the same performance as C/C++
- Memory safe
- With great interoperability
- Outstanding tooling

Source: `https://cheats.rs/`

Rust is...

Good because

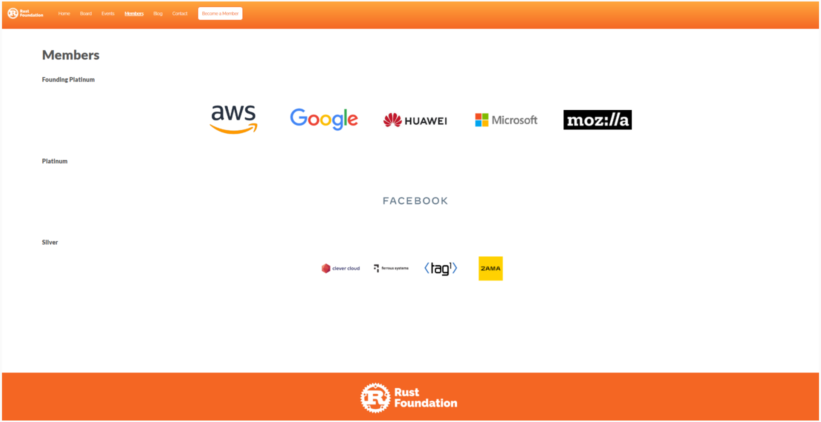
- A compiled, strongly typed, systems programming language (thing C/C++)
- Fast - about the same performance as C/C++
- Memory safe
- With great interoperability
- Outstanding tooling

Bad because

- Steep learning curve
- Long compile times
- A somewhat young language

Source: `https://cheats.rs/`

The Rust Foundation



```
fn main() {
        // the type is not necessary in this case
        let x: f64 = 1_000_000.0;

        // no type cast - it is infered by the compiler
        let y = {
                let z = x / 100.0;
                // notice the missing colon
                // this is the return value of the block
                z
        };

        println!("The value of y is {}", y);
}
```

One more Hello World

```rust
fn main() {
        // this variable we will want to change
        let mut x: f64 = 0.0;

        for i in 1..=10 {
                // we perform a cast, as i is not f64
                x += i as f64;
        }
        // we recreate the value x to remove mutability
        let x = x;
        // we take the address of the variable x
        let y = &x;

        // we pass a reference, but still get what we want
        println!("The sum of the first 10 natural numbers is {}", y);
}
```

One more Hello World 2

```
struct MyStruct {
        has_value: bool,
        large_vector: Vec<f64>,
}

fn freeFlowFunction(x: &[f64]) -> f64 {
        unimplemented!();
}

impl MyStruct {
        // method functions for the struct
        fn increate_x(&mut self) {
                self.x += 1;
        }
}

impl fmt::Debug for MyStruct {
        // implement a trait for MyStruct - in this case,
        // how to present the struct in debug messages
        fn fmt(&self, f: &mut fmt::Formatter<'_>) -> fmt::Result {
                todo!()
                // almost never need to do this:
                // annotate the struct definition with
                // #[derive(Debug)]  and the compiler will handle it
        }
}
```

One more Hello World 2f2f aka Family

```
fn a_fun(x: String) {
        if x.len() % 2 == 0 {
                println!("Even");
                return;
        }
        println!("Odd");

}
fn main() {
        let x = "Summer School 2021".to_string();
        a_fun(x);
        println!("x was: '{}' ", x);
}
```

One more Hello World 2f2f aka Family

```rust
fn a_fun(x: String) {
        if x.len() % 2 == 0 {
                println!("Even");
                return;
        }
        println!("Odd");

}
fn main() {
        let x = "Summer School 2021".to_string();
        a_fun(x);
        println!("x was: '{}' ", x);
}
```

```
error[E0382]: borrow of moved value: `x`
   --> src\main.rs:701:26
    |
699 |     let x = "Summer School 2021".to_string();
    |         - move occurs because `x` has type `String`, which does not implement the `Copy` trait
700 |     a_fun(x);
    |           - value moved here
701 |     println!("x was: '{}'", x);
    |                             ^ value borrowed here after move

error: aborting due to previous error; 1 warning emitted
```

Figure 1: ”Not a bad way to explain the error in the code”

Rules of ownership

1. Each object can be used only once - it is "moved" every time
2. Out of scope objects are destroyed and cannot be used
3. Each block produces a value that goes one scope up
4. All objects have a lifetime that constrains which scopes they may be moved out of

Source: `https://chrismorgan.info/blog/rust-ownership-the-hard-way/`

Rules of ownership

1. Each object can be used only once - it is ”moved” every time
2. Out of scope objects are destroyed and cannot be used
3. Each block produces a value that goes one scope up
4. All objects have a lifetime that constrains which scopes they may be moved out of

There are always exceptions to any rule...

Source: `https://chrismorgan.info/blog/rust-ownership-the-hard-way/`

Rules of ownership

1. Each object can be used only once - it is ”moved” every time
2. Out of scope objects are destroyed and cannot be used
3. Each block produces a value that goes one scope up
4. All objects have a lifetime that constrains which scopes they may be moved out of

There are always exceptions to any rule...

1. Object that implement the special trait *Copy* (think as if on every ”=” we do a copy)
2. Reference borrowing - you can have as many as you like immutable references to an object or one mutable

Source: `https://chrismorgan.info/blog/rust-ownership-the-hard-way/`

1. Functional tools (could possibly be faster than loops)

```
let sum_squares = a_vec.iter().map(|v| v * v).sum();
```

2. Algebraic types

```
enum IpAddr {
        V4(u8, u8, u8, u8),
        V6(String),
}
```

3. Destructuring and matching

```
let as_string = match ip {
        IpAddr::V4(a, b, c, d) => format!("{}.{}.{}.{}", a, b, c, d),
        IpAddr::V6(s) => s.clone()
};
```

4. Anyone can implement a Trait for their Struct

Two important Enums

- ```rust
  enum Result<T, E> {
          Ok(T),
          Err(E),
  }
  ```
  You either get the value or get an error - Rust doesn't have exceptions that can get swept under the rug.
- ```rust
  enum Option<T> {
          None,
          Some(T),
  }
  ```
  None is only part of this enum, again forcing you to handle missing values.

Memory safety, concurrency

- The ownership, type system and the borrow checker are some of what guarantees the memory safety in Rust. The compiler proves that no Segmentation Faults, memory leaks etc. cannot happen - otherwise, you get compile time error.
- "Fearless concurrency" - the same checks are done when multiple threads are involved, so two threads accessing the same memory location with read and write is not possible (mostly).
- If that was all, Rust would be too rigid and unusable, hence

  ```
  unsafe
  ```

  code word. This signals to the compiler: "I, the human, checked this piece of code and it is fine." You are free to do whatever in these sections (e.g. pointer arithmetic).

Tooling

Some example tools for development (non exhaustive)

- `rustc` - the compiler (already saw the error messages)
- `rustup` - updater of the above
- `cargo` - package manager (crate in rust parlance)
- `cargofmt` - formatter (so the code looks nice)
- `clippy` - linter (helps find potential issues)
- `rust-analyzer` - IDE support (i.e. code completion) can integrate into your favorite text editor

Motivating Example Revisited

Extending Python

- Python C Package - write directly C code (i.e. extend CPython)
- Cython - mixture of Python and C (C++) syntax - this is parsed to C and then compiled as above
- With enough dedication any language that can build a dynamic library (and some C files to be link the two) can be used for Python modules (e.g. GoLang)

- Python C Package - write directly C code (i.e. extend CPython)
- Cython - mixture of Python and C (C++) syntax - this is parsed to C and then compiled as above
- With enough dedication any language that can build a dynamic library (and some C files to be link the two) can be used for Python modules (e.g. GoLang)

Enter PyO3:

- A Rust bindings for Python
- Most of the 'magic' happens with the help of annotations
- Several different ways to actually build the package.

```rust
#[pyclass]
struct ScaledTModel {
        data_size: usize,
        nu: f64,
        tau2: f64,
        mu: f64,
        alpha2: f64,

        extended_vars: Vec<f64>,
        data: Vec<f64>,
        result_mu: Vec<f64>,
        result_sigma2: Vec<f64>,
}
#[pymodule]
fn rust_great_model(_py: Python, m: &PyModule) -> PyResult<()> {
        m.add_class::<ScaledTModel>()?;

        Ok(())
}
```

Exposed methods

```rust
#[pymethods]
impl ScaledTModel {
        #[new]
        fn new(data: Vec<f64>, nu: f64) -> Self {
                println!("making a rust model");
                ScaledTModel {
                        // skipped in slides!
                }
        }

        #[getter]
        fn get_mu(&self) -> PyResult<Vec<f64>> {
                Ok(self.result_mu.clone())
        }

        #[getter]
        fn get_sigma2(&self) -> PyResult<Vec<f64>> {
                Ok(self.result_sigma2.clone())
        }

        fn run(&mut self, burn_in: usize,
                        sample_size: usize) -> PyResult<()> {
                todo!()
        }
}
```

Folder Structure

```
great_model
│   Cargo.lock                    <---- Rust related
│   Cargo.toml                    <---- Rust related
│   MANIFEST.in                   <---- Some Rust changes
│   pyproject.toml
│   requirements.txt
│   setup.py                      <---- Some Rust changes
│
├───great_model
│   ├───numpy_great_model
│   │       scaled_t_model.py
│   │       __init__.py
│   │
│   └───python_great_model
│           scaled_t_model.py
│           __init__.py
│
└───src
    lib.rs                        <---- Rust related (source code)
```

Cargo.toml

```toml
[package]
name = "great_model"
version = "1.0.0"
edition = "2018"

# See more keys and their definitions at
# https://doc.rust-lang.org/cargo/reference/manifest.html

[lib]
name = "great_model"
# "cdylib" is necessary to produce a shared
# library for Python to import from.
#
# Downstream Rust code (including code in `bin/`, `examples/`,
# and `tests/`) will not be able the code unless the "rlib" or
# "lib" crate type is also included, e.g.:
# crate-type = ["cdylib", "rlib"]
crate-type = ["cdylib"]

[dependencies]
rand = "0.8.4"
rand_distr = "0.4.1"
rand_xoshiro = "0.6.0"

[dependencies.pyo3]
version = "0.14.1"
features = ["extension-module"]
```

MANIFEST and pyproject

A MANIFEST.in file consists of commands, one per line, instructing setuptools to add or remove some set of files from the distribute package .

```
include pyproject.toml Cargo.toml
recursive-include src *
```

pyproject.toml specifies the minimum build system for python (basically - make setup.py work)

```
[build-system]
requires = [
        "setuptools>=42",
        "wheel",
        "setuptools-rust"
]
build-backend = "setuptools.build_meta"
```

setup.py

```
1  import setuptools
2  from setuptools_rust import Binding, RustExtension
3
4  setuptools.setup(
5          name="great-model",
6          version="1.0.0",
7          author="Metodi Nikolov",
8          author_email="metodi.nikolov@gmail.com",
9          classifiers=[
10                 "Programming Language :: Python :: 3",
11                 "License :: OSI Approved :: MIT License",
12                 "Operating System :: OS Independent",
13         ],
14         rust_extensions=[RustExtension("great_model.rust_great_model",
15                                         binding=Binding.PyO3)],
16         zip_safe=False,
17         packages=setuptools.find_packages(),
18         python_requires=">=3.8",
19 )
```

Lines 2, 14-16 are need for the rust bindings, the rest are standard python package.

Implementing the model

```
fn run(&mut self, burn_in: usize, sample_size: usize) -> PyResult<()> {
        let mut rng = Xoshiro256Plus::seed_from_u64(0);
        for _ in 0..burn_in {
                self.update_extended_vars(&mut rng);
                self.update_alpha2(&mut rng);
                self.update_mu(&mut rng);
                self.update_tau2(&mut rng);
        }

        self.result_mu.resize(sample_size, 0.0);
        self.result_sigma2.resize(sample_size, 0.0);

        for i in 0..sample_size {
                self.update_extended_vars(&mut rng);
                self.update_alpha2(&mut rng);
                self.update_mu(&mut rng);
                self.update_tau2(&mut rng);
                self.result_mu[i] = self.mu;
                self.result_sigma2[i] = self.alpha2 * self.tau2;
        }
        Ok(())
}
```

Implementing the model

```
fn update_mu<R>(&mut self, rng: &mut R)
        where
                R: Rng,
{
        let mut tmp: f64;
        let mut variance = 0.0;
        let mut expected_value = 0.0;
        for (datum, ext_var) in self.data.iter().
                        zip(self.extended_vars.iter()) {
                tmp = 1.0 / ext_var;
                variance += tmp;
                expected_value += datum * tmp;
        }
        variance /= self.alpha2;
        expected_value /= self.alpha2;
        variance = 1.0 / variance;
        expected_value = expected_value * variance;

        self.mu = expected_value + variance.sqrt() *
                        rng.sample::<f64, _>(StandardNormal);
}
```

```rust
fn update_tau2<R>(&mut self, rng: &mut R)
        where
                R: Rng,
{
        let x: f64 = self.extended_vars.iter().map(|x| 1.0 / x).sum();
        let scale = 2.0 / (self.nu * x);
        let gamma = Gamma::new(
                (self.data_size as f64) * self.nu / 2.0,
                scale,
                ).expect(&format!(
                        "Failed at creation of Gamma, shape:{:}, scale: {:}",
                        (self.data_size as f64) * self.nu / 2.0,
                        2.0 / (self.nu * x)
                ),
        );
        self.tau2 = gamma.sample(rng);
}
```

Implementing the model

```
fn update_alpha2<R>(&mut self, rng: &mut R)
        where
                R: Rng,
{
        let mu = self.mu;
        let mut x = self
                .data
                .iter()
                .zip(self.extended_vars.iter())
                .map(|(datum, ext_var)| (datum - mu) * (datum - mu) / ext_var)
                .sum();
        x /= self.data_size as f64;
        self.alpha2 = sample_scaled_inv_chi_square(rng, self.data_size as _, x);
}
```

Implementing the model

```
fn update_extended_vars<R>(&mut self, rng: &mut R)
        where
                R: Rng,
{
        let mut x: f64;
        let mu = self.mu;
        let alpha2 = self.alpha2;
        let tau2 = self.tau2;
        let nu = self.nu + 1.0;

        let nutau2 = self.nu * tau2;
        let chi = ChiSquared::new(nu).expect(
                                        "Failed at creation of ChiSquared");
        for i in 0..self.data_size {
                let datum = self.data[i];
                x = (datum - mu) * (datum - mu) * alpha2;
                self.extended_vars[i] = (nutau2 + x) / chi.sample(rng);
        }
}
```

Rust implementation

```
from great_model.rust_great_model import ScaledTModel as RustModel
import numpy as np
from timeit import timeit

nu = 6
mu = -3.14
sigma2 = 30
data_size = 500
z = np.random.randn(data_size)
x = np.random.chisquare(nu, data_size)

t_data = mu + z * np.sqrt(sigma2 * nu / x)

r_model = RustModel(t_data, nu)

exec_time = timeit('r_model.run(2000, 2000)',
        globals=globals(), number = 10)
print(f"Average run time of Rust model: {exec_time * 1000 / 10:3.5f} ms")
Average run time of Rust model: 113.34160 ms
```

Next Steps

Leaving you with..

- Language `https://www.rust-lang.org/`
- The Book `https://doc.rust-lang.org/stable/book/`
- Good introduction `https://tourofrust.com/`
- Reference (work in progress) `https://doc.rust-lang.org/stable/reference/`
- Reddit `https://www.reddit.com/r/rust/`
- `https://github.com/MetodiNikolov/SummerSchool2021`

Thank you for the attention!