Retour d'expérience sur la mise en ligne de simulations : Shiny et Voilà

Daphné Giorgi (IR CNRS, LPSM – Sorbonne Université)

16 octobre 2023

Journées Mathrice 2023
IHP, Paris

Motivations

Motivations

Situation de départ: Plusieurs collègues ont des simulations

- didactiques
- de recherche

But: Mise en ligne des simulations.

Points d'attention:

- Plusieurs languages : R, C++, Python
- Les collègues n'ont pas de connaissances en développement web
- Moi non plus !
- Le support informatique change
- Les religions aussi !

Première version avec Flask

Première version : Flask

Flask : un microframework écrit en Python.

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

hello.py

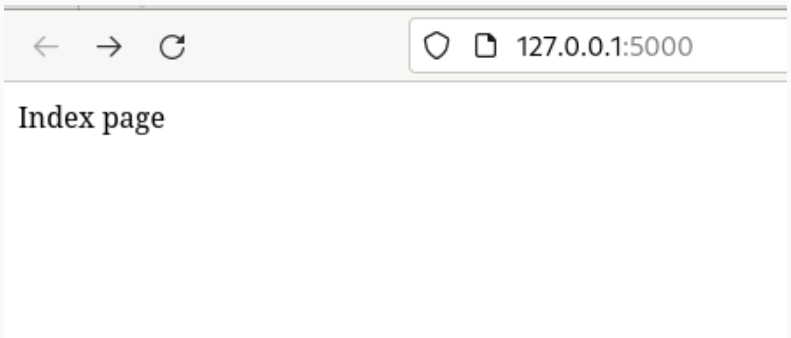
```
> python hello.py
$ flask --app hello run
 * Serving Flask app 'hello'
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
```

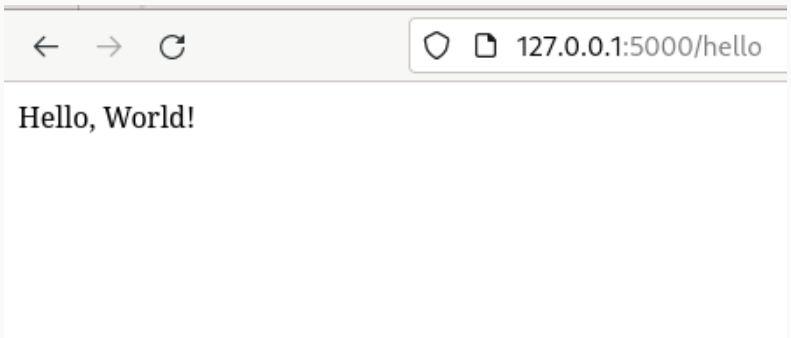
Accès intuitif

Routage : Le décorateur `route()` permet la définition d'URL significatives.

```python
@app.route('/')
def index():
    return 'Index Page'

@app.route('/hello')
def hello():
    return 'Hello, World'
```





Formulaires

WTForms : bibliothèque Python qui permet un rendu flexible des formulaires web.

```python
from flask_wtf import FlaskForm
from wtforms import IntegerField
from wtforms.validators import DataRequired

class MyForm(FlaskForm):
    N = IntegerField('Size', validators=[DataRequired()])
```

simulation_form.py

Templates

jinja2 : Moteur de templates. Ensemble de fichiers (html) qui contiennent des variables et des expressions qui seront remplacées par des valeurs au moment du rendu du template.

```
{% from "_macros.jinja2" import render_field}

...

<form>
    {{ render_field(form.N) }}
</form>

...
```

simulation.html

Templates

Rendu de templates : Appel de rendu.

```python
from flask import render_template

@app.route('/simulation')
def simulation():
    # Just render the form
    form = MyForm()
    return render_template("simulation.html", form=form)
```

main.py

Requêtes GET et POST

Méthodes HTTP : L'argument `methods` du décorateur `route()` permet de gérer différentes méthodes HTTP.

```python
from flask import request

@app.route('/simulation', methods=['GET', 'POST'])
def simulation():
    if request.method == 'GET':
        # Just render the form
        # ...
    else :
        # Compute simulation with form data and show result
        # ...
```

main.py

Templates

jinja2 : Moteur de templates. Ensemble de fichiers (html) qui contiennent des variables et des expressions qui seront remplacées par des valeurs au moment du rendu du template.

```
{% from "_macros.jinja2" import render_field}

<script src="{{ url_for('render_script', filename='simulation.js') }}">
</script>
...
<form>
    {{ render_field(form.N) }}
</form>
...
<button id="button_compute">COMPUTE</button>

<div id="output"></div>
```

simulation.html

Javascript

Javascript : Lancement de la simulation et affichage du résultat.

```
function compute(event){
     $.post('{{ url_for("simulation") }}', form, function(data){
         content = fct_html(data)
         $('#output').html(content)
         }, 'json');
}

function register_callbacks(){
    // Attaches behavior to UI components.
    $("body").on("click",  "#button_compute", compute);
}

$(document).ready(function(){
    // "main" function
    register_callbacks();
});
```

simulation.js

Rendu final

```python
from flask import request
from flask import render_template

@app.route('/simulation', methods=['GET', 'POST'])
def simulation():
    if request.method == 'GET':
        # Just render the form
        form = MyForm()
        return render_template("simulation.html", form=form)
    else :
        form = MyForm(request.form)
        # simulate fct in simulation.py file
        response = simulate(form) #json
        return json.dumps(response)
```

main.py

Arborescence

```
main.py
|
|__forms
|    |__ simulation_form.py
|
|__modules
|    |__ simulation.py
|
|__templates
.    |__ simulation.html
.    |__ simulation.js
.
```

Langages différents

Simulations en C++ : Utilisation de `SWIG` pour wrapper le code `C++` en module Python.

Simulation en R : Utilisation de R Shiny dans un `iframe`.

Deuxième version : Voilà et Shiny

Shiny

````
---
title: "Test Shiny"
date: "16 novembre 2023"
output: html_document
runtime: shiny
---
Test Latex: \[f(x) = \sum_{i=1}^n x^n\]

```{r, echo=FALSE}
inputPanel(
  selectInput("n_breaks", label = "Number of bins:",
              choices = c(10,15,20, 35, 50), selected = 20)
)

renderPlot({
  hist(faithful$eruptions, probability = TRUE, breaks = as.numeric(input$n_breaks),
       xlab = "Duration (minutes)", main = "Geyser eruption duration")

  dens <- density(faithful$eruptions, adjust = 1)
  lines(dens, col = "blue")
})
```
````

short_test.Rmd

Test Shiny

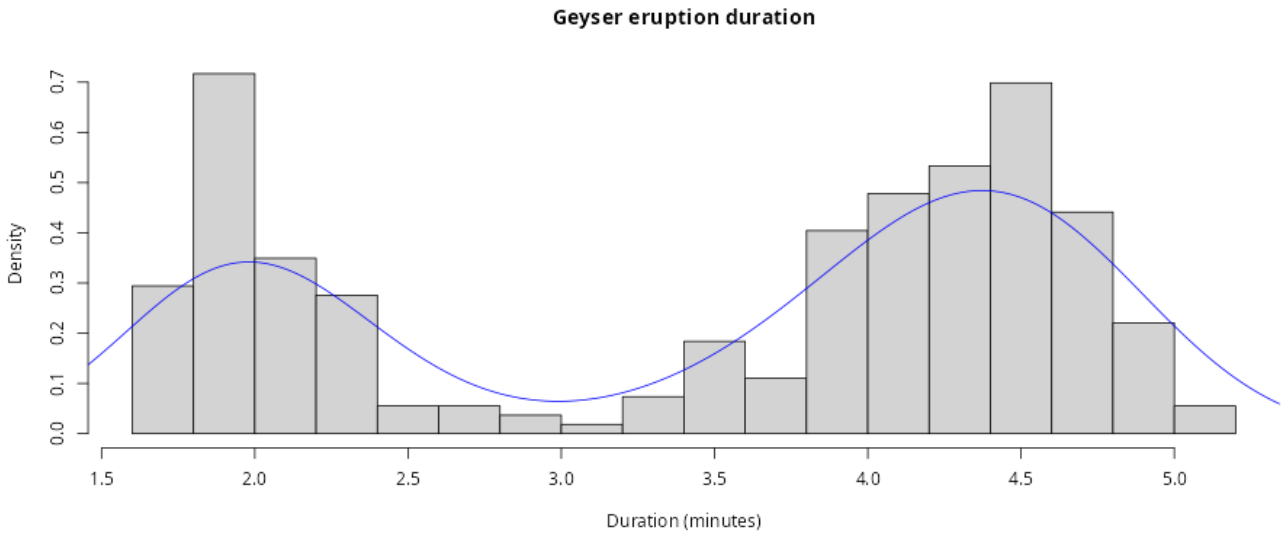
16 novembre 2023

Test Latex:

$$f(x) = \sum_{i=1}^{n} x^n$$

Number of bins:

20



Shiny tutorial and examples

Voilà

https://github.com/voila-dashboards/voila

Voila crée un rendu des notebooks Jupyter avec des widgets interactifs.

- Voilà execute le code d'un notebook et collectionne les outputs.
- Le notebook et ses résultats sont convertis en HTML. Par défaut, les cellules de code du notebook sont cachées.
- Cette page est servie soit en tant qu'application Tornado, soit via le serveur Jupyter.
- Lorsque les utilisateurs accèdent à la page, les widgets de la page ont accès au noyau Jupyter.

Voilà exemple

So easy, *voilà*!

In this example notebook, we demonstrate how Voilà can render Jupyter notebooks with interactions requiring a roundtrip to the kernel.

Jupyter Widgets

[ ]:
```
import ipywidgets as widgets

slider = widgets.FloatSlider(description='$x$')
text = widgets.FloatText(disabled=True, description='$x^2$')

def compute(*ignore):
    text.value = str(slider.value ** 2)

slider.observe(compute, 'value')

slider.value = 4

widgets.VBox([slider, text])
```

Basic outputs of code cells

[ ]:
```
import pandas as pd

iris = pd.read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv')
iris
```

voila_test.ipynb

Voilà exemple

So easy, *voilà*!

In this example notebook, we demonstrate how Voilà can render Jupyter notebooks with interactions requiring a roundtrip to the kernel.

Jupyter Widgets

x 4.00

x^2 16

Basic outputs of code cells

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

http://localhost:8866/

Nouvelle arborescence

```
index.html
|
|__shiny
|    |__ simulation.Rmd
|
|__voila
|    |__ simulation.ipynb
|
|__swig
```

Site

En cours de migration:

https://simulations.lpma.math.upmc.fr/