Coding as a Service: librería pypsdier, aprendizajes y metodología.

pycon Argentina 2020

Sebastian Flores

16 al 27 de Noviembre de 2020

Coding as a Service: librería pypsdier, aprendizajes y metodología.

Observaciones:

- sebastiandres en twitter y github.
- Opiniones y críticas a título personal.
- Presentación realizada con la extensión RISE, fácilmente reproducible.
- Presentación disponible en https://www.github.com/sebastiandres/charlas (https://www.github.com/sebastiandres/charlas)

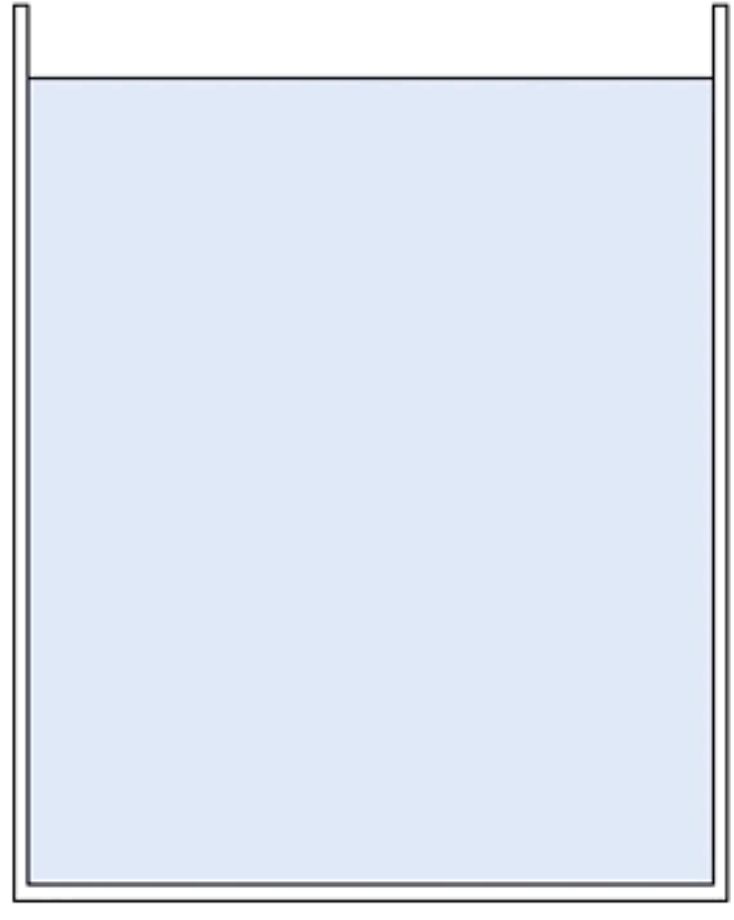
Observación a los "reviewers": el video será grabado usando la interactividad de la extensión RISE. La conversión a pdf no representa la presentación de manera fidedigna. De ser posible, revisar esta versión en línea:
https://github.com/sebastiandres/charlas/tree/master/2020_11_XX_pycon_ar_pypsdier (https://github.com/sebastiandres/charlas/tree/master/2020_11_XX_pycon_ar_pypsdier)

Coding as a Service: librería pypsdier, aprendizajes y metodología.

Introducción

- ¿Cómo distribuir y documentar un proyecto?
- ¿Cómo desarrollar y mantener código de manera sencilla?
- ¿Cómo colaborar con otras personas que no son expertos en computación?

El Problema Inicial



El Problema Inicial

Inicialmente, buscábamos resolver para un radio R fijo y conocido. El problema es obtener $S_b(t)$ y $S(t, r)$, para $t > 0$ y $0 < r < R$.

$$\frac{\partial S}{\partial t}(t, r) = D_S \left(\frac{\partial^2 S}{\partial r^2}(t, r) + \frac{2}{r} \frac{\partial S}{\partial r}(t, r) \right) - V\left(S(t, r)\right)$$

Condición de borde en el centro de las partículas:

$$\frac{\partial S}{\partial r}(t, 0) = 0 \text{ para } t > 0$$

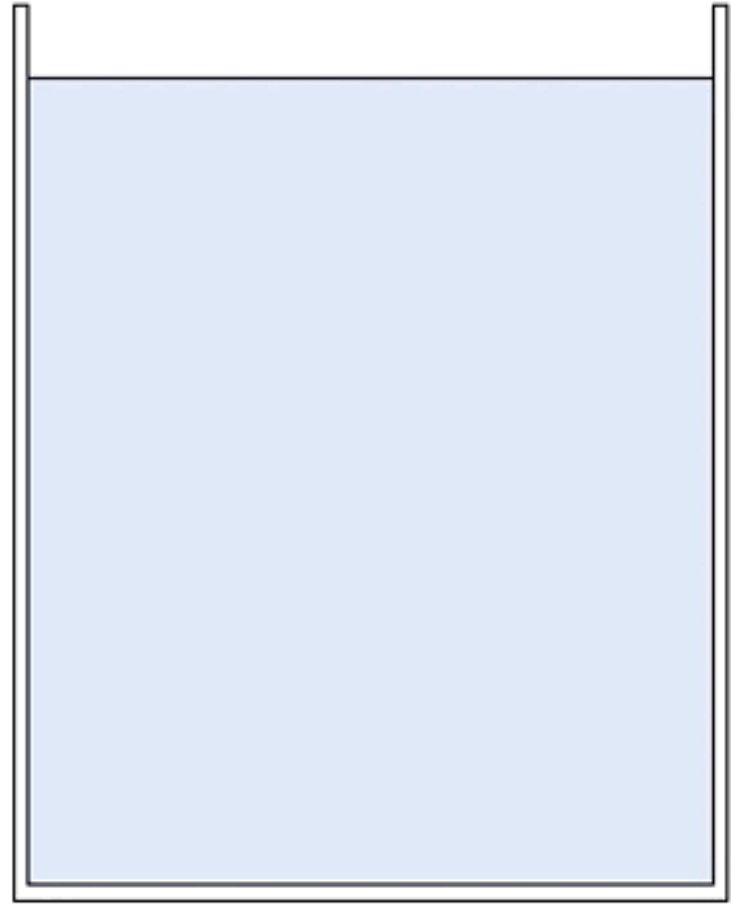
Condiciones de borde en la superficie de la partícula:

$$S_b(t) = S(t) \text{ para } t > 0$$
$$\frac{dS_b}{dt}(t) = -3D_S \frac{V_c}{V_R R} \frac{\partial S(t, R)}{\partial R} \text{ para } t > 0$$

Con la condición inicial

$$S_b(0) = S_0$$
$$S(0, r) = 0 \text{ para } 0 \leq r < R$$

El Problema Inicial



El Problema Actual

El problema anterior se generalizó considerando partículas de distinto radio. El problema es obtener $S_b(t)$ y $S(t, r, R_i)$, para $t > 0, 0 < r < R_i$.

$$\frac{\partial S}{\partial t}(t, r, R_i) = D_S \left(\frac{\partial^2 S}{\partial r^2}(t, r, R_i) + \frac{2}{r} \frac{\partial S}{\partial r}(t, r, R_i) \right) - V\left(S(t, r, R_i)\right) I(t) Z(r, R_i)$$

Condición de borde en el centro de las partículas:

$$\frac{\partial S}{\partial r}(t, 0, R_i) = 0 \text{ para } t > 0$$

Condiciones de borde en la superficie de la partícula:

$$S_b(t) = S(t, R_i, R_i) \text{ para } t > 0$$
$$\frac{dS_b}{dt}(t) = -3 D_S \frac{V_c}{V_R E\left[R^3\right]} E\left[R^2 \frac{\partial S}{\partial r}\bigg|_{r=R} \right] \text{ para } t > 0$$

Con la condición inicial

$$S_b(0) = S_0$$
$$S(0, r, R_i) = 0 \text{ para } 0 \leq r < R_i$$

Coding as a Service: librería pypsdier, aprendizajes y metodología.

Los Desafíos

Desafíos técnicos:

- ¿Cómo discretizar las ecuaciones?
- ¿Cómo realizar la implementación numérica?

Desafíos organizacionales:

- ¿Cómo colaborar?
- ¿Cómo distribuir el código?
- ¿Cómo simplificar el uso?
- ¿Como hacer reproducible las simulaciones?

Coding as a Service: librería pypsdier, aprendizajes y metodología

Historia

- Primeras implementaciones en matlab.
    - Problema: Cada nueva reacción química requiere una nueva implementación numérica.
    - Repetición de código, mantención tediosa y reiterativa.
- Desarrollo en python + numpy: pypsdier
    - Código genérico. Número arbitrario de sustancias (sustratos o productos) y reacciones definidas .
    - Tamaño de particulas definidas por una lista de radios y probabilidad asociada.
    - Interface simple e instalación por pypi.

El impacto del desarrollo conjunto es actualmente de 10 publicaciones.

Coding as a Service: librería pypsdier, aprendizajes y metodología

Framework: ¿Qué necesitábamos?

- Propocionar instalación y versionamiento de python, jupyter y librerías necesarias.
- Simplificar el deployment y versionamiento de un código.
- Exponer una interface simple al usuario final, permitiendo ocultar una implementación numérica compleja y con una curva de aprendizaje apropiado.
- Permitir almacenar y compartir resultados de simulación, para que sean analizados, reproducidos y almacenados.
- Permitir el uso de recursos computacionales en la nube.
- Permitir una buena documentación para desarrolladores y usuarios.

Coding as a Service: librería pypsdier, aprendizajes y metodología

Framework: Propuesta

- **Libraría instalable por pip**: instalación desde pypi o github/bithub/otro, intalarlo en local o en mybinder/colab.
- **Versionamiento por git y versionamiento de librerías**: desarrollo incremental y que facilita colaboración.
- **Documentación por read-the-doc**: para usuarios finales y desarrolladores.
- **Interface simple mediante orientación a objetos** para simplificar y hacer más amigable.
- **Semilla de simulación**: contiene toda la información de la simulación (inputs, configuración del sistema y librerías, opciones y outputs). Esta semilla puede ser almacenada y compartida.

Coding as a Service: librería pypsdier, aprendizajes y metodología

Ejemplo

Primero que nada, instalaremos la librería:

In [ ]:
```
#!pip install -i https://test.pypi.org/simple/ Pypsdier
!pip install git+https://github.com/sebastiandres/pypsdier.git
```

Ejemplo

Consideremos un ejemplo de una reacción simple. Definamos los inputs de la simulación:

In [ ]:

```
def MichaelisMenten(S, E0, k, K):
  """Definition for Michaelis Menten reaction with inputs E0 [mM], k [1/s] and K
 [mM]"""
  return (-k*E0*S[0]/(K+S[0]), )

inputs = {}
inputs["SimulationTime"] = 1*60. # [s]
inputs["SavingTimeStep"] = 10. # [s]
inputs["CatalystVolume"]  = 0.01 # [mL]
inputs["BulkVolume"]  = 40.0  # [mL]
inputs["Names"] = ('Substrat',)  # legend for the xls, reports and plots
inputs["InitialConcentrations"] = (1.0,)   # [mM]
inputs["EffectiveDiffusionCoefficients"] = (1.0E-9,)  # [m2/s]
inputs["CatalystParticleRadius"] = [100.E-6] # [m]
inputs["CatalystParticleRadiusFrequency"] = [1.0] # []
inputs["ReactionFunction"] = MichaelisMenten # function
inputs["ReactionParameters"] = (41 , 0.13)   # [1/s], [mM/s], parameters
inputs["CatalystEnzymeConcentration"] = 0.5 # [mM]s
```

Ejemplo

Definamos ahora opciones de gráficos para la simulación:

In [ ]:
```
plot_options = {}
plot_options["xlabel"] = "Tiempo de reacción [s]"
plot_options["ylabel"] = "Concentración [mM]",
plot_options["title" ] = "Simulación de Michaelis Menten",
plot_options["data_x"] = [ 0.1, 2.1,  3.9,  6.1,  7.9,  9.9],
plot_options["data_y"] = [-2.8, 3.6, 10.7, 13.6, 22.8, 27.1],
plot_options["data_kwargs"] = {'label':'exp', 'color':'red',
                           'marker':'s', 'markersize':6,
                           'linestyle':'none','linewidth':2}
plot_options["sim_kwargs"] = {'label':'sim', 'color':'black',
                           'marker':'o', 'markersize':6,
                           'linestyle':'dashed','linewidth':2}
```

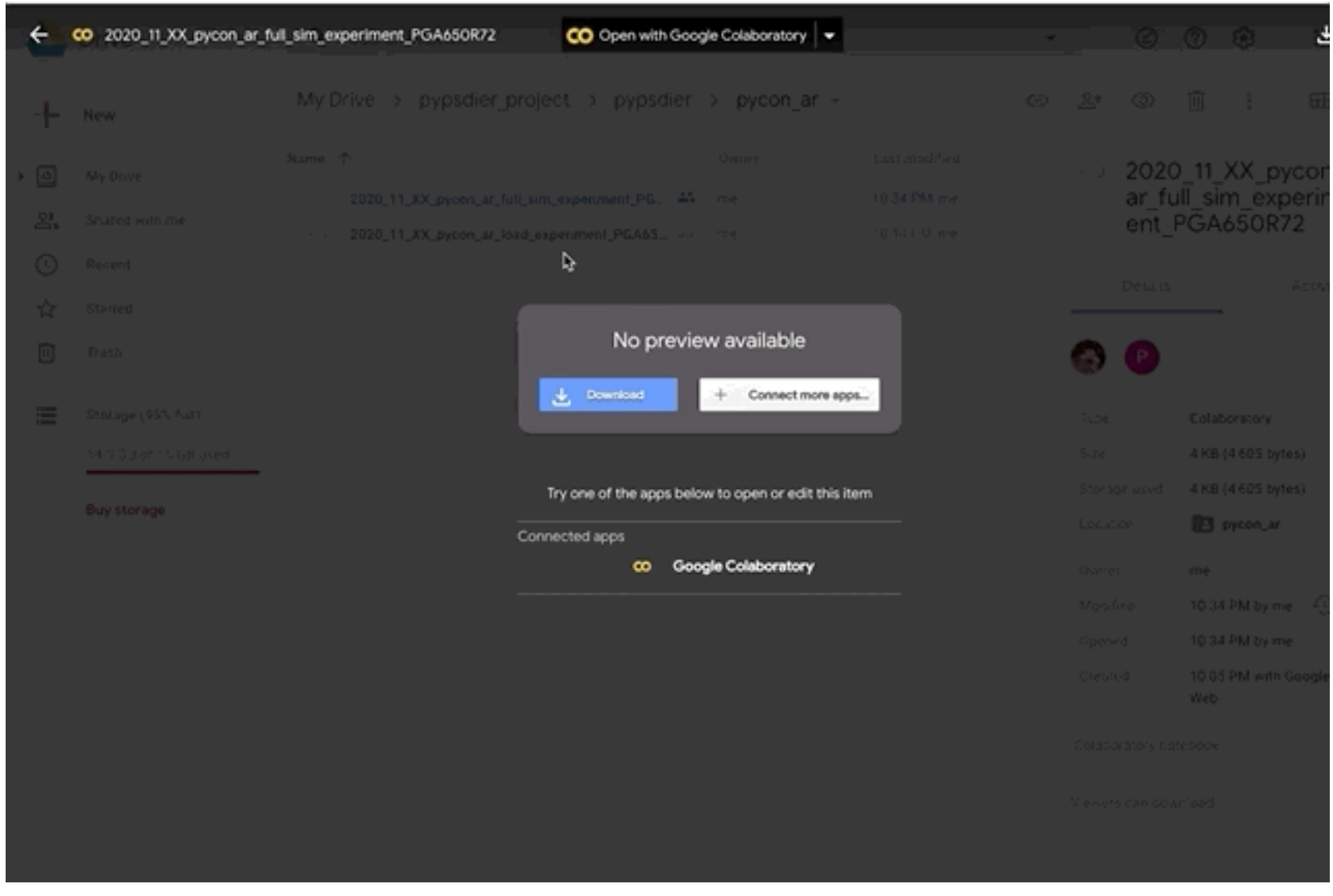
Ejemplo

Todo está listo para realizar la simulación:

In [ ]:
```
import pypsdier
SI = pypsdier.SimulationInterface()
SI.new(inputs, plot_options)
SI.simulate("ode")
SI.simulate("pde")
SI.save("example_0.rde")
#SI.plot(filename="example_0.png")
SI.export_xls("example_0.xls")
```

Reproducibilidad

Cuando se desarrolla una librería instalable por pypi, es posible realizar simulaciones sin siquiera tener que instalar python en el computador: se pueden realizar en mybinder o Google Colab, como en el siguiente enlace: https://bit.ly/3mD51hn (https://bit.ly/3mD51hn)


2020_11_XX_pycon_ar_full_sim_experiment_PGA650R72
Open with Google Colaboratory
New
My Drive
Shared with me
Recent
Starred
Trash
Buy storage
My Drive > pypsdier_project > pypsdier > pycon_ar
No preview available
Download
Connect more apps...
Try one of the apps below to open or edit this item
Connected apps
Google Colaboratory
Details
Colaboratory
4 KB (4 605 bytes)
pycon_ar
me
10 34 PM by me
10 34 PM by me

Reproducibilidad

Cuando se desarrolla una librería instalable por pypi, es posible realizar simulaciones sin siquiera tener que instalar python en el computador: se pueden realizar en mybinder o Google Colab, como en el siguiente enlace: https://bit.ly/3mD51hn (https://bit.ly/3mD51hn)

2020_11_XX_pycon_ar_full_sim_experiment_PGA650R72

File Edit View Insert Runtime Tools Help

Comment Share

+ Code + Text

RAM Disk Editing

```
[3] inputs["InitialConcentrations"] = (10.0, 0., 0.)   # [mM], initial concentration of substrates and products
    inputs["EffectiveDiffusionCoefficients"] = (5.30E-10, 7.33E-10, 5.89E-10)  # [m2/s], effective diffusion coefficient
    inputs["CatalystParticleRadius"] = (75.7E-6,) # [m], list of possible catalyst particle radiuses
    inputs["CatalystParticleRadiusFrequency"] = (1.0,) # [], list of corresponding frequencies of catalyst particle radius
    inputs["ReactionFunction"] = PenG_HydrolysisReaction # function defining the reaction
    inputs["ReactionParameters"] = 41., 0.13, 821., 1.82, 48.  #[1/s] and [mM]*4  # [1/s], [mM/s], parameters to be used
    inputs["CatalystEnzymeConcentration"] = 0.140 # [mM] can be a float, int or a function returning float or int.

    plot_options = {}
    plot_options["t_exp"] = [0,  2,  4,  6,  8, 10, 20, 30, 40, 50, 65, 80, 100, 120, 150] # Time in mins
    plot_options["PenG_exp"] = [10.00, 9.68, 9.30, 8.93, 8.85, 8.70, 8.00, 6.50, 5.21, 3.59, 2.58, 1.65, 1.01, 0.49, 0.25

    # Simulate only if file not found
    SI = pypsdier.SimulationInterface()
    SI.new(inputs, plot_options)
```

¿Qué versiones de todas las librerías necesarias estamos usando? ¿Qué datos serán utilizados en la simulación?

```
SI.status()
```

Simulemos y guardemos el resultado:

```
[ ] # Define filename for storing the simulation and plots
    filename = "PGA650R72.rde"

    SI.simulate("ode")
    SI.simulate("pde")
    SI.save(filename)
```

Framework Propuesto

El framework se ha disponibilizado facilitar que un cierto código pueda disponibilizarse. Para ello basta con hacer un fork del repositorio https://github.com/sebastiandres/GenericSimulationLibrary (https://github.com/sebastiandres/GenericSimulationLibrary) y actualizar:

- En `simulation_interface.py` actualizar los métodos: new, load, simulate, save, plot, export.
- En `docs/source/*.rst` actualizar la documentación.

Conclusión

- Herramientas actuales hacen sencilla la colaboración interdisciplinaria.
- El framework propuesto simplifica el desarrollo y la reproducibilidad del código.
- Python + numpy permite una implementación numérica flexible y robusta.

Cierre

Las librerías están disponibles para descarga o instalación:

- Librería `pypsdier`: https://github.com/sebastiandres/pypsdier (https://github.com/sebastiandres/pypsdier)
- Framework `Coding as a Service`: https://github.com/sebastiandres/GenericSimulationLibrary (https://github.com/sebastiandres/GenericSimulationLibrary)

Gracias por sus comentarios.

Les agradecería responder una encuesta sobre la charla:

https://bit.ly/2ZBOPTT