RadicalPy

Radical pair spin dynamics for the masses

Lewis M. Antill[1,2] and Emil Vatai[3]
lmantill.github.io & **vatai.github.io**

1. Graduate School Science and Engineering, Saitama University
2. JST PRESTO（Quantum Biology）
3. High Performance Artificial Intelligence Systems Research Team, RIKEN Center for Computational Science

Motivation

- The radical pair mechanism has gained popularity in recent years:
  - Biological magnetoreception
- Spin chemistry is a relatively niche area of science and can be daunting for people trained in different fields.
- Current toolboxes, such as EasySpin[1] and Spinach[2] focus on EPR and NMR spin dynamics simulations that require the proprietary commercial[3] software MATLAB.

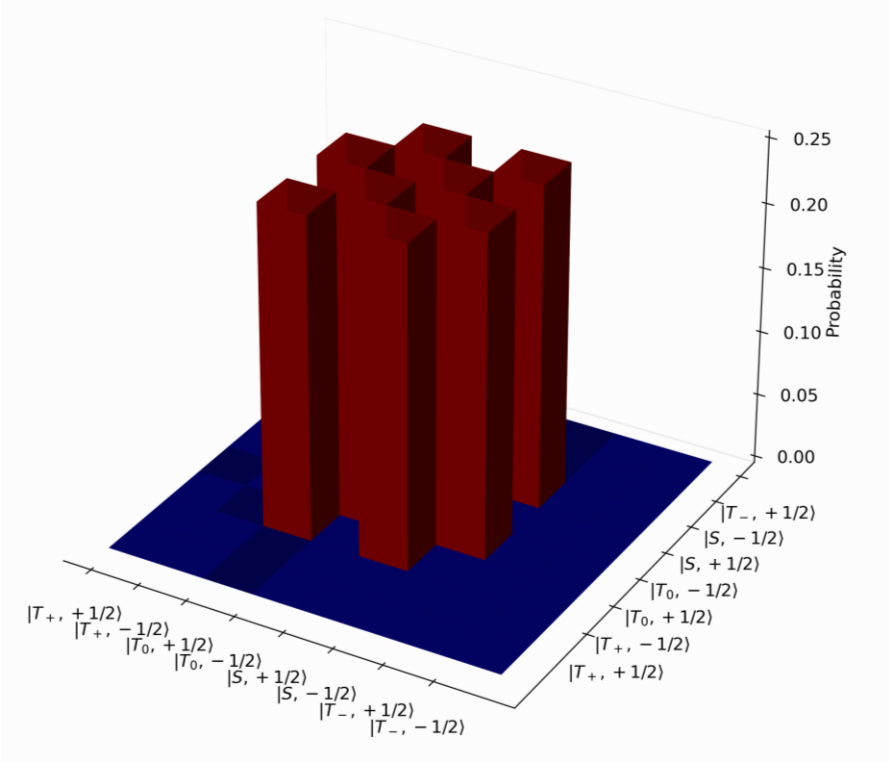
- We aim to build an intuitive (object-oriented) open-source framework in the Python programming language (which is freely available) specific to radical pair spin dynamics.
- Designed for specialists and non-specialists alike to learn and be able perform a plethora of spin dynamics simulations with relative ease.

1. S. Stoll, A. Schweiger, *J. Magn. Reson.,* 178(1), 42-55 (2006)
2. H. J. Hogben, M. Krzystyniak, G. T. P. Charnock, P. J. Hore, I. Kuprov, *J. Magn. Reson.*, 208, 179-194 (2011)
3. Non-negligible price $$$$$

RadicalPy

- Open-source, written in Python, available on GitHub:
  - https://github.com/Spin-Chemistry-Labs/radicalpy
  - `python -m pip install radicalpy`
  - https://radicalpy.readthedocs.io/
- Tutorials
  - Jupyter Notebook (coming soon)
- CI/CD (automated tests, automated documentation)
- License: MIT License
- Standard Python packages:
  - NumPy (http://numpy.scipy.org/)
  - SciPy (http://www.scipy.org/)
  - Matpotlib (http://matplotlib.sourceforge.net/)
- 4,236 lines of code (so far)
- Still in development

Time evolution of a 3D density matrix of a 1-proton singlet born radical pair (with automated spin state labels)



github://examples/density_matrix_3d.py

L. M. Antill* and E. Vatai*, RadicalPy: Radical pair spin dynamics for the masses, *in preparation*

RadicalPy

data.py

Databases:
- Physical constants
- Pauli matrices
- Molecules
  - HFCs
  - Effective HFCs
- Isotopes
  - Spin multiplicities
  - Magnetogyric ratios

simulation.py

- Monte Carlo random walk:
  - Solution-based RPs
  - Encapsulated RPs
- Kinetics
  - Relaxation mechanism
- Molecular dynamics interface

HilbertSimulation

LiouvilleSimulation

- Basis: Zeeman and ST
- Spin states (S, T_+, T_-, T_0, T_{+-}, Equilibrium)
- Spin operators
- Product operators
- Unitary propagators
- Hamiltonians (Zeeman, Hyperfine, …)
- Time evolution
- MARY
- MIE
- Anisotropy

classical.py

kinetics.py

- Exponential model
- Haberkorn
- Jones-Hore

relaxation.py

- ST-dephasing
- TT-dephasing
- Random fields
- g-tensor anisotropy
- T1 relaxation
- T2 relaxation

plot.py

- Density matrix
- Energy level
- Linear energy level
- 3D polar plot
- 3D random walk

estimations.py

- J and D
- τ_C
- Various rates
- $B_{1/2}$

utils.py

- Conversions
- $B_{1/2}$ fitting
- Rotation matrices
- Autocorrelations
- Spectral densities

Building molecules:
Interface to molecule and isotope databases

```
from radicalpy.simulation import Molecule
```

Using the molecule database:

```
flavin = Molecule(radical="flavin_anion",
                  nuclei=["H25"])
print(flavin)
Molecule: flavin_anion
  HFCs: [array([[ 0.47570582,  0.04192216, -0.00921916],
       [ 0.04192216,  0.4626044 , -0.00827001],
       [-0.00921916, -0.00827001,  0.42560999]])]
  Multiplicities: [2]
  Magnetogyric ratios (mT): [267522.18744]
```

Using the isotope database:

```
Molecule(nuclei=["1H", "14N"],
         hfcs=[0.41, 1.82])
Molecule: N/A
  HFCs: [0.41, 1.82]
  Multiplicities: [2, 3]
  Magnetogyric ratios(mT): 267522.18744,
                           19337.792]
  Number of particles: 2
```

Helpful error messages:

```
Molecule("adenine_cation", ["buz"])
    Traceback (most recent call last):
    ...
    ValueError: Available nuclei below.
    N6-H2 (hfc = -0.66)
    N6-H1 (hfc = -0.63)
    C8-H (hfc = -0.55)
```

Custom molecules:

```
Molecule("my_flavin", multiplicities=[2],
          gammas_mT=[267522.18744],
          hfcs=[0.5])
Molecule: my_flavin
  HFCs: [0.5]
  Multiplicities: [2]
  Magnetogyric ratios(mT):[267522.18744]
  Number of particles: 1


Z = Molecule("kryptonite")
Molecule: kryptonite
  HFCs: []
  Multiplicities: []
  Magnetogyric ratios (mT): []
  Number of particles: 0
```

Building simulators:
Laying the groundwork for simulations

```
import radicalpy as rp

flavin = rp.simulation.Molecule("flavin_anion", ["H25"])
Z = rp.simulation.Molecule("Z")
sim = rp.simulation.HilbertSimulation([flavin, Z])

print(sim)

Number of electrons: 2
Number of nuclei: 1
Number of particles: 3
Multiplicities: [2, 2, 2]
Magnetogyric ratios (mT): [-176085963.023, -176085963.023, 267522.18744]
Nuclei: ['H25']
Couplings: [0]
HFCs (mT): [array([[ 0.47570582,  0.04192216, -0.00921916],
       [ 0.04192216,  0.4626044 , -0.00827001],
       [-0.00921916, -0.00827001,  0.42560999]])]
```

```
H = sim.total_hamiltonian(B=0, D=0, J=0)

plt.spy(H)
plt.show()
```

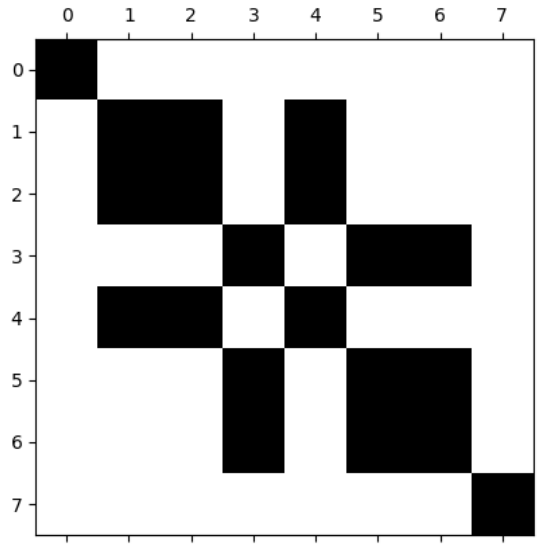
$$H_Z = \gamma_e B_0 [\hat{S}_{Lx} \sin\theta\cos\phi + \hat{S}_{Ly} \sin\theta\sin\phi + \hat{S}_{Lz} \cos\theta]$$

$$H_H = \sum_k^{N_L} [\hat{S}_L \cdot A_{Lk} \cdot \hat{I}_{Lk}] \qquad L = \{A, B\}$$

$$H_J = -J\left(2\hat{S}_A \cdot \hat{S}_B + \frac{1}{2}\hat{E}\right) \qquad \hat{E} = \hat{P}^S + \hat{P}^T$$

$$H_D = \sum_{j=x,y,z} \sum_{k=x,y,z} D_{jk} \hat{S}_{Aj} \hat{S}_{Ak}$$

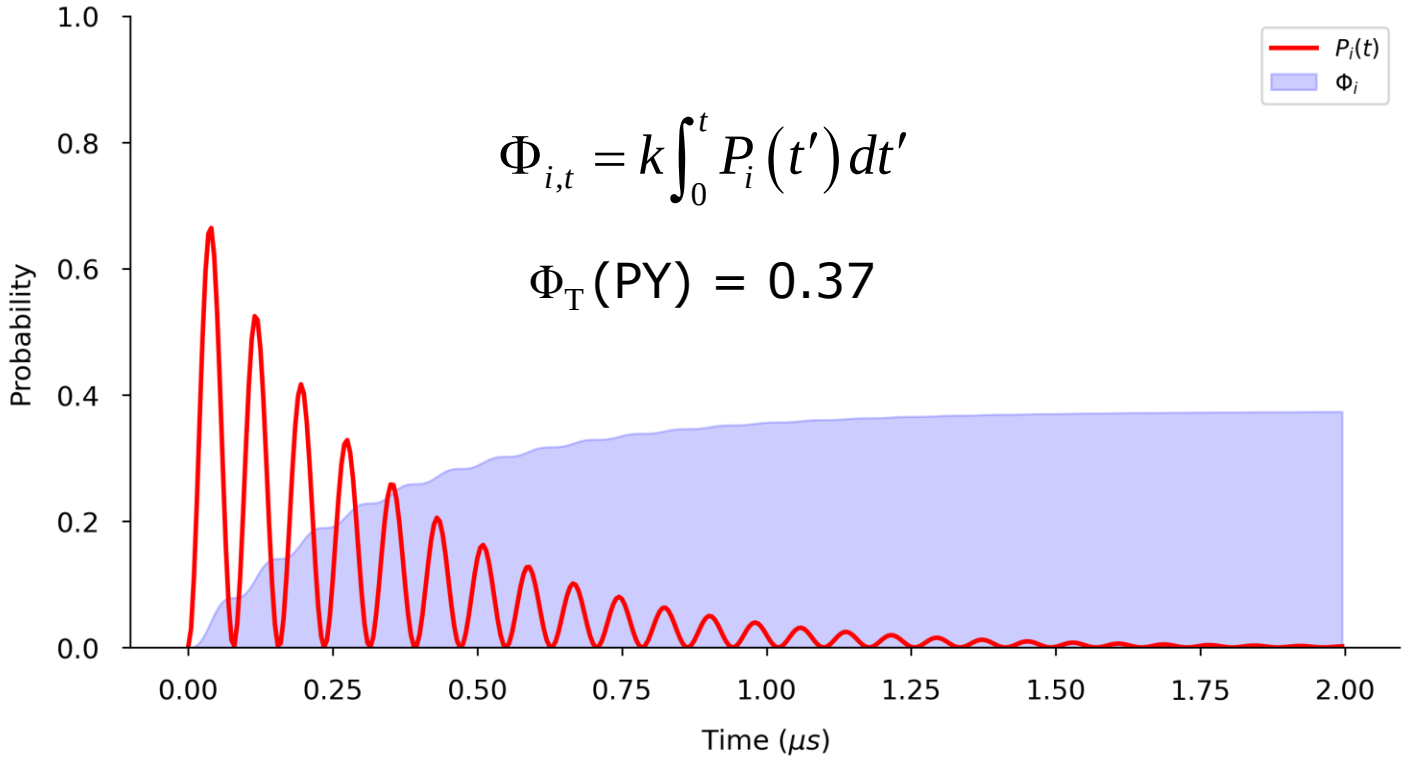
$$H = H_Z + H_H + H_J + H_D$$



Time evolution and product yield

Hilbert space simulation:
Time evolution and product (triplet) yield for a singlet born flavin-Z radical pair

```
time = np.arange(0, 2e-6, 5e-9)
rhos = sim.time_evolution(State.SINGLET, time, H)

k = 3e6
kinetics=[rp.kinetics.Exponential(k)]

time_evol = sim.product_probability(State.TRIPLET,
                                    rhos)
sim.apply_hilbert_kinetics(time, time_evol, kinetics)
product_yield, product_yield_sum =
          sim.product_yield(time_evol, time, k)

x = time * 1e6

plt.plot(x, time_evol, color="red", linewidth=2)
plt.fill_between(x, product_yield,
                 color="blue", alpha=0.2)
plt.xlabel("Time ($\mu s$)")
plt.ylabel("Probability"); plt.ylim([0, 1])
plt.legend([r"$P_i(t)$", r"$\Phi_i$"])

print(f"PY = {product_yield_sum}")
```



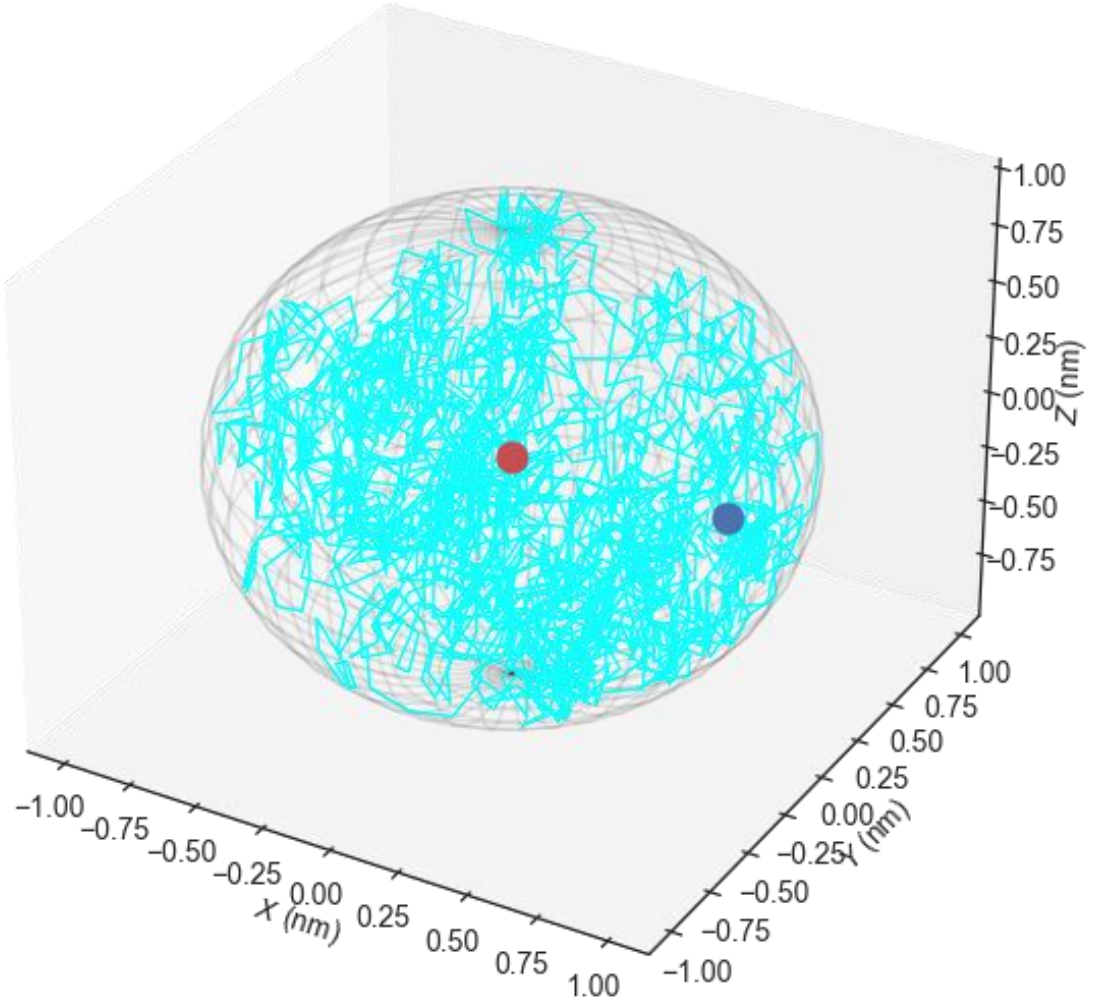
Monte Carlo random walk: Encapsulated radical pair motion

Diffusional motion of the radical pair can be considered as random jumps between two points separated by distance Δr[1],

$$\Delta r = \sqrt{6 D_{AB} \Delta t}$$

Δt = time interval
D_{AB} = mutual diffusion coefficient of the radical pair

```
from radicalpy import classical

n_steps = 2000
r_min = 0.5e-9 / 2
r_max = 2e-9 / 2
r = (r_min) + np.random.sample() * ((r_max) - (r_min))
x0, y0, z0 = r, 0, 0
mut_D = 1e-6 / 10000
del_T = 40e-12

delta_r = classical.get_delta_r(mut_D, del_T)
pos, dist, ang = classical.randomwalk_3d(n_steps, x0, y0,z0,
                                         delta_r, r_max)
plot.monte_carlo_caged(pos, r_max)
```



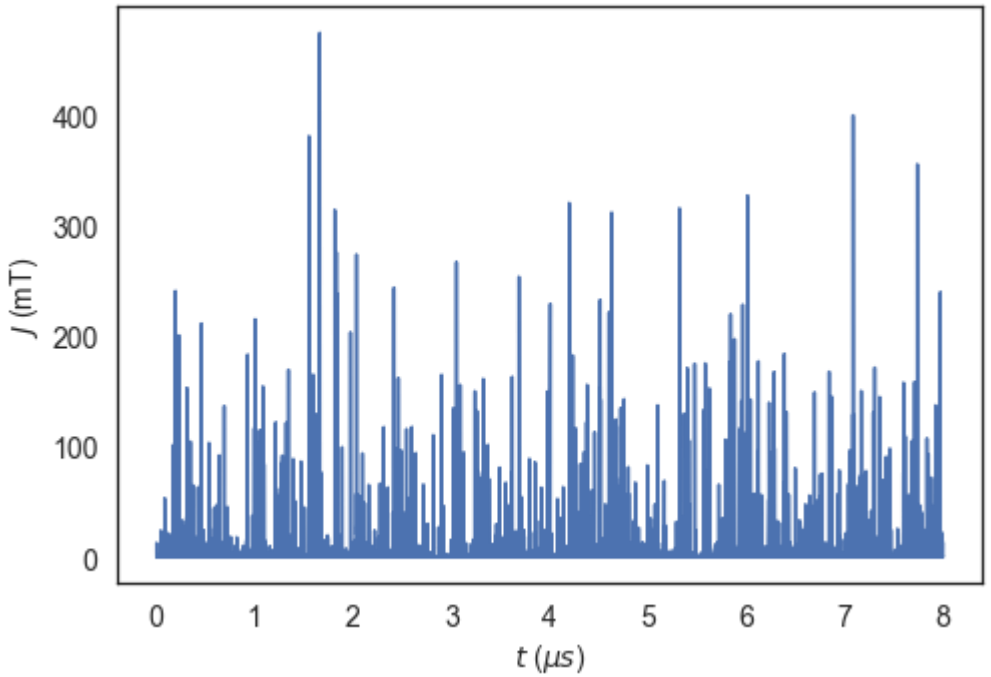
2 nm diameter micelle / vesicle

1. T. Miura and H. Murai, *J. Phys. Chem. B*, 119, 5534-5544 (2015)

Correlation time estimation

Exchange interaction[1,2]: $J(r) = J_0 e^{-\alpha(r(t)-r_{\min})}$

```
t, r, J, D =
classical.monte_carlo_exchange_dipolar(n_steps -1,
          r_min, del_T, pos[:,0], dist[0:-1], ang)

fig = plt.figure()
...
```

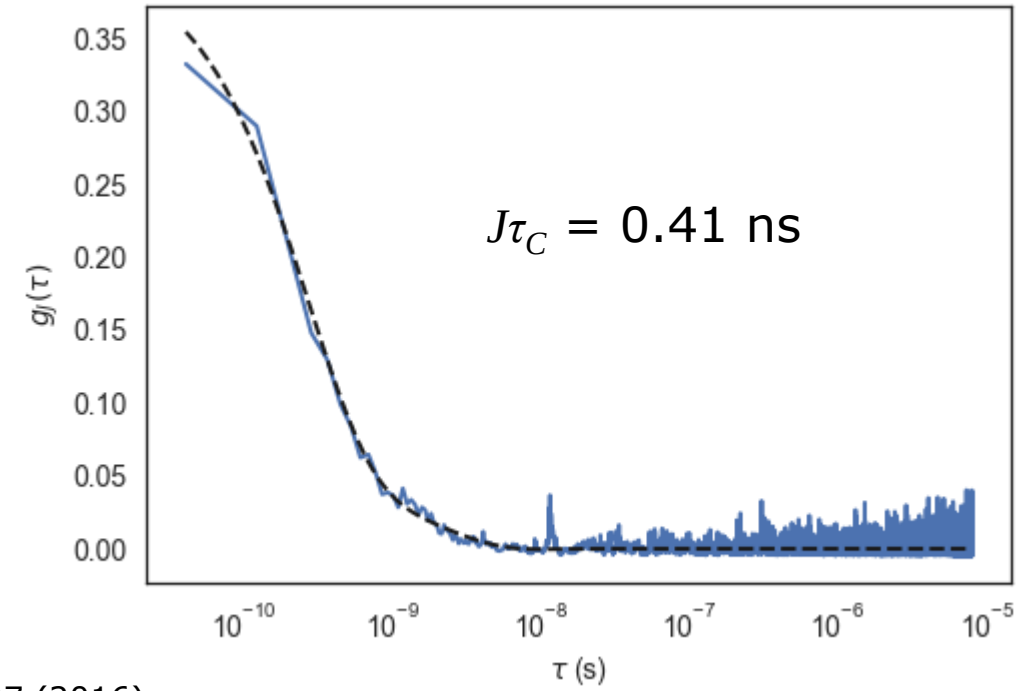


Autocorrelation function[2]:

$$g_J(\tau) = \left\langle \left[J(t) - J_{av} \right] \left[J(t+\tau) - J_{av} \right] \right\rangle$$

From 100 exponential fits we can estimate the correlation time τ_C

```
acf_j = rp.utils.autocorrelation(J, factor=2)

acf_j_popt, acf_j_pcov = curve_fit(
    rp.utils.multiexponential_fit, x2, y2, p0=a + taus)

acf_j_As, acf_j_taus, tauc_j =
rp.estimations.correlation_time_from_fit(*acf_j_popt)
```



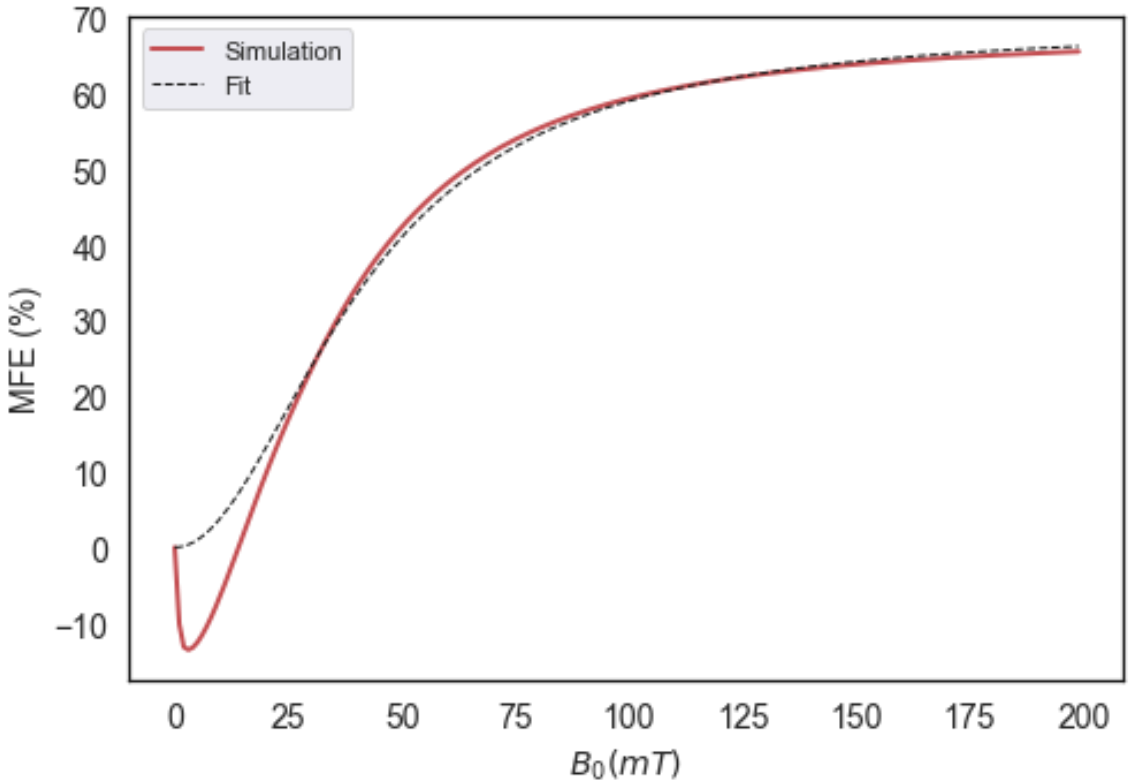
1. T. Miura and H. Murai, *J. Phys. Chem. B*, 119, 5534-5544 (2015)
2. D. R. Kattnig, J. K. Sowa, I. A. Solov'yov, and P. J. Hore, *New J. Phys.* 18, 063007 (2016)

ST-dephasing rate estimation and MARY

Using Redfield theory one can obtain[1]:

$$k_{STD} = 4\left\langle\left[\left(J\left(r\left(t\right)\right)-J_{av}\right)^{2}\right]\right\rangle\tau_{C} = 1.33\times10^{9}\,s^{-1}$$

```
kSTD = rp.estimations.k_STD(J, tauc_j)


rftb = rp.simulation.Molecule("flavin_anion", ["N5", "H25"])
indole = rp.simulation.Molecule("tryptophan_cation",
                                ["H7", "Hbeta1"])
sim = rp.simulation.LiouvilleSimulation([rftb, indole])
time = np.arange(0, 15e-6, 5e-9)
Bs = np.arange(0, 200, 1)

results = sim.MARY(
    init_state=State.TRIPLET,
    obs_state=State.TRIPLET,
    time=time,
    B=Bs,
    D=0,
    J=0,
    kinetics=[
        kinetics.Haberkorn(8.4e6, State.SINGLET),
        kinetics.HaberkornFree(1.6e6),
    ],
    relaxations=[relaxation.SingletTripletDephasing(kSTD)],
)

Bhalf, x_model_MARY, y_model_MARY, MARY_fit_error, R2 =
                        rp.utils.Bhalf_fit(Bs, results["MARY"])
```



HFE = 65.56 %

LFE = -13.54 %

$B_{1/2}$ = 41.36 mT

$B_{1/2}$ fit error = 0.73 mT

R^2 for $B_{1/2}$ fit = 0.979

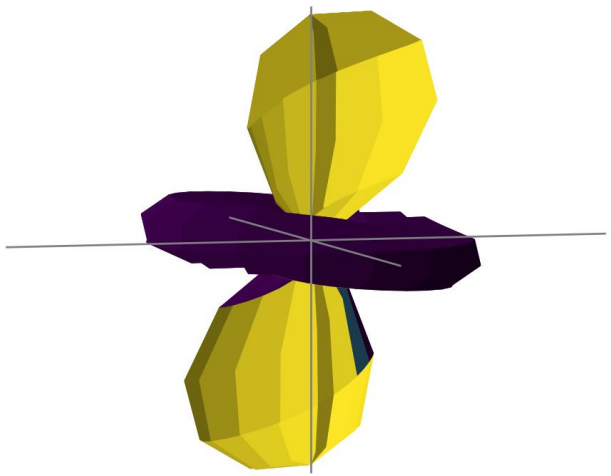
1. D. R. Kattnig, J. K. Sowa, I. A. Solov'yov, and P. J. Hore, *New J. Phys.* 18, 063007 (2016)

Radical pair anisotropy

```
theta = np.linspace(0, np.pi, 17)
phi = np.linspace(0, 2 * np.pi, 32)

flavin = rp.simulation.Molecule("flavin_anion", ["N5", "N10"])
Z = rp.simulation.Molecule("zorro", [])
sim = rp.simulation.HilbertSimulation([flavin, Z])

time = np.arange(0, 5e-6, 5e-9)
B0 = 0.05
k = 1e6

results = sim.anisotropy(
    init_state=State.SINGLET,
    obs_state=State.SINGLET,
    time=time,
    theta=theta,
    phi=phi,
    B=B0,
    D=0,
    J=0,
    kinetics=[rp.kinetics.Exponential(k)],
)

Y = results["product_yield_sums"]
delta_phi_s, gamma_s = rp.utils.yield_anisotropy(Y,
                                    theta, phi)
Y = Y - rp.utils.spherical_average(Y, theta, phi)

rp.plot.anisotropy_surface(theta, phi, Y)
```

$$\Phi_S^{anis}(\theta,\phi) = \Phi_S(\theta,\phi) - \langle \Phi_S \rangle$$

$$\langle \Phi_S \rangle = \frac{1}{4\pi} \int_0^{2\pi} \int_0^{\pi} \Phi_S(\theta,\phi) \sin\theta d\theta d\phi$$



$$\Delta\Phi_S = \max \Phi_S - \min \Phi_S = 0.262$$

$$\Gamma = \frac{\Delta\Phi_S}{\langle \Phi_S \rangle} = 0.702$$

O. Efimova and P. J. Hore, *Biophysical Journal*, 94, 1565–1574 (2008)
D. R. Kattnig, I. A. Solov'yov, and P. J. Hore, *Phys. Chem. Chem. Phys.*, 18, 12443 (2016)

Outlook

simulation.py

- Spherical tensor basis
- Two-site model
- RYDMR
- Field switching (SEMF)
- Time-resolved MARY
- Semi-classical
- EPR – SCRP
- Coherent control

relaxation.py

- Rotationally modulated relaxation (Bloch-Redfield-Wangsness theory)

utils.py

- Wigner functions
- Irreducible spherical tensors
- Gaussian / Orca parser

kinetics.py

- Diffusion model (Noyes)

Democratising radical pair spin dynamics will require a community:

- Comments
- Remarks
- New molecules (json files)
- Pull requests (expanding the code base)

Future work:

- Large scale (more accurate) spin dynamics simulations on supercomputers (*e.g.*, Fugaku)

Acknowledgements

Supervisors
Jonathan R Woodward (Tokyo)
Kiminori Maeda (Saitama)

Collaborator
Peter J Hore (Oxford)

Help with correlation time estimation
Daniel Kattnig (Exeter)

Funding

国立研究開発法人
科学技術振興機構
Japan Science and Technology Agency

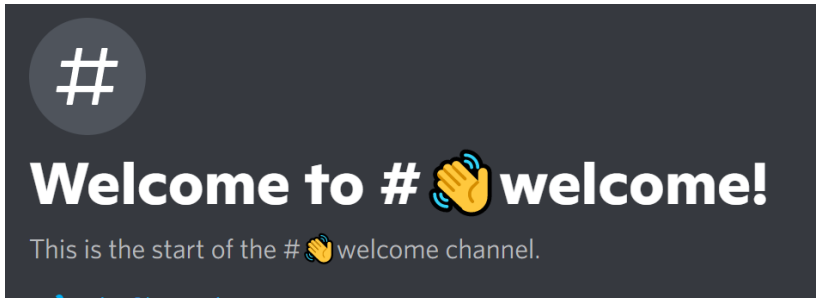
さきがけ
PRESTO

科研費
KAKENHI

Spin Chemistry Community

https://spin-chemistry-community.github.io/

https://twitter.com/SpinChemistry



https://discord.io/spin-chemistry-community/