MESolver

J. F. Coker

April 22, 2020

Abstract

This document shows the equations used by the program MESolver and how they are implemented in-code.

Once the list of sites, the transfer integrals between them, and the simulation parameters have been read from provideded input files, the rate matrix $\boldsymbol{A}$, can be found. For M particles, this is an $M \times M$ matrix with elements

$$A_{i,j} = \begin{cases} \Gamma_{j,i}, & i \neq j \\ -\sum_{i \neq j} \Gamma_{i,j}, & i = j \end{cases},$$

where $\Gamma_{i,j}$ is the charge transfer rate from site i to site j. For a system of two sites this will look like

$$\boldsymbol{A} = \begin{pmatrix} -\Gamma_{1,2} & \Gamma_{2,1} \\ \Gamma_{1,2} & -\Gamma_{2,1} \end{pmatrix}.$$

In-code this is implemented by the following section:

```
std::cout << "\nCreating rate matrix A...\n";
gsl_matrix* A = gsl_matrix_alloc(M, M);
gsl_matrix_set_zero(A);
for (int i = 0; i < M; i++)
  for (int f = 0; f < M; f++)
  {
    if (i == f)
    {
      double sum = 0.0;
      for (int k = 0; k < M; k++)
        if (i != k)
          sum += allSites[i].Rate(&allSites[k], F_z, kBT, reorg);
      gsl_matrix_set(A, i, f, -sum);
    }
    else
      gsl_matrix_set(A, i, f, allSites[f].Rate(&allSites[i], F_z, kBT, reorg));
  }
```

The rates themselves are calculated from Semi-classical Marcus Theory, using the equation

$$\Gamma_{i,f} = \frac{2\pi}{\hbar} |J_{i,f}|^2 (4\pi\lambda k_B T)^{-\frac{1}{2}} \exp\left(-\frac{(\Delta E + \lambda)^2}{4\lambda k_B T} \right),$$

where $J_{i,f}$ is the transfer integral between i and j, λ is the reorganisation energy, T is the temperature and ΔE is the energetic driving force. This is implemented in-code by the following method:

```
double site::Rate(site* pSite, double fieldZ, double kBT, double reorg)
{
  neighbour* pN = this->hasNeighbour(pSite);

  if (!pN)
    // Sites aren't interacting, transfer rate will be zero.
    return 0.0;
  else
  {
  double J2 = std::pow(pN->_J,2); // |J_if|^2
  double deltaE = this->deltaE(pSite, fieldZ);
  return ((2 * pi) / hbar) * J2 * std::pow(4 * pi * reorg * kBT, -0.5) * std::exp(-1
      * std::pow(deltaE + reorg, 2) / (4 * reorg * kBT));
  }
}
```

This calls another method which calculates ΔE using

$$\Delta E = \Delta\varepsilon + q\left(\boldsymbol{r}\cdot\boldsymbol{F}\right),$$

where $\Delta\varepsilon = (\varepsilon_f - \varepsilon_i)$ is the difference in site energies, q is the elementary charge, $\boldsymbol{r} = (\boldsymbol{r}_f - \boldsymbol{r}_i)$ is the displacement between the site centers-of-mass and $\boldsymbol{F}$ is the applied electric field. This method is implemented as:

```
double site::deltaE(site* pSite, double fieldZ)
{
  return (pSite->energy - this->energy) + (pSite->pos.Z - this->pos.Z) * fieldZ;
}
```

Once the matrix $\boldsymbol{A}$ is set up, we are ready to solve the master equation using singular value decompostion...