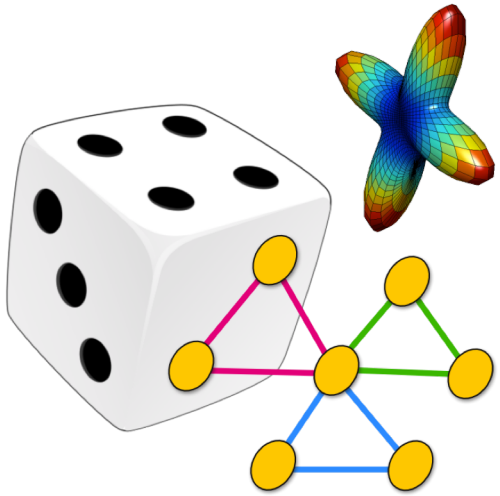


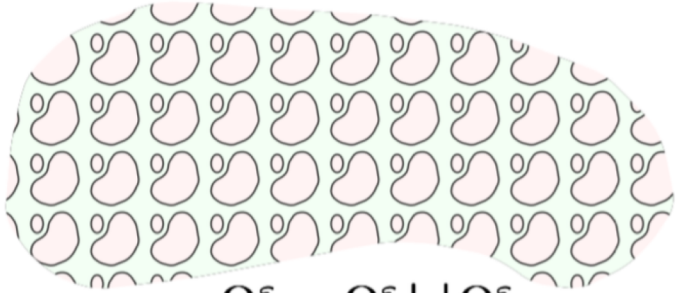
Scientific computing in Magnetic Resonance Imaging

Simona Schiavi
DICE LAB, Diffusion Imaging and Connectivity Estimation
Computer Science Department, Univerisity of Verona

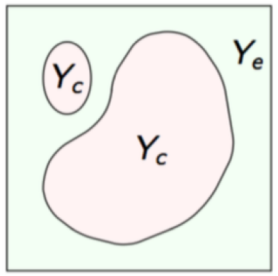
Course schedule

- May 31st, 8:30-10:30, alpha: Introduction to python
- June 07th, 8:30-10:30, alpha: How to obtain RM images and python lab (FFT)
- June 10th, 10:30-12:30, H: Bloch Torrey equation and homogenization techniques
- June 11th, 8:30-10:30, gamma: solution of Bloch Torrey equations in simple 2D geometry in FreeFem
- June 12th, 14:30-15:30, F: Numerical Convex Optimization applied to diffusion MRI

Bloch-Torrey Equation



$\Omega^\varepsilon = \Omega_e^\varepsilon \bigcup \Omega_c^\varepsilon$



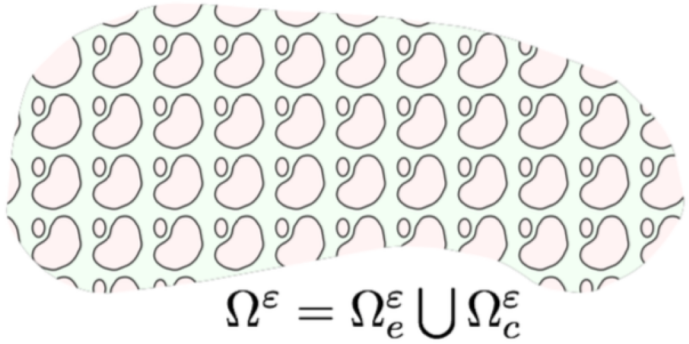
Scaled periodic cell Y,
Zoom $\sim 1/\varepsilon$

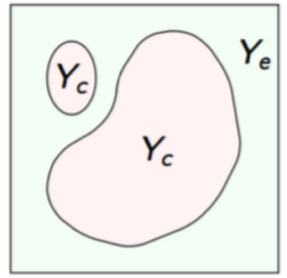
$$\overline{\Omega_e^\varepsilon} = \bigcup_{\xi \in \Xi_\varepsilon} \varepsilon(\xi + \overline{Y_e}) \cap \overline{\Omega}, \quad \Omega_c^\varepsilon = \bigcup_{\xi \in \Xi_\varepsilon} \varepsilon(\xi + Y_c) \cap \Omega, \quad \Omega_{ext}^\varepsilon = \Omega_e^\varepsilon \cup \Omega_c^\varepsilon.$$

$$\partial\Omega_c^\varepsilon \cap \partial\Omega_e^\varepsilon = \partial\Omega_e^\varepsilon \setminus \overline{\partial\Omega} = \bigcup_{\xi \in \Xi_\varepsilon} \Gamma_m^{\varepsilon,\xi}, \quad \text{where} \quad \Gamma_m^{\varepsilon,\xi} = \varepsilon(\xi + \Gamma_m) \cap \Omega.$$

$$\begin{cases} \dfrac{\partial}{\partial t} M_\varepsilon(\mathbf{x},t) + \iota \mathbf{q} \cdot \mathbf{x} f(t) M_\varepsilon(\mathbf{x},t) - \operatorname{div}(\hat{\mathcal{D}}_{0\varepsilon}(\mathbf{x}) \nabla M_\varepsilon(\mathbf{x},t)) = 0 & \text{in } \Omega^\varepsilon \times]0, T_E[\\ \hat{\mathcal{D}}_{0\varepsilon} \nabla M_\varepsilon \cdot \nu|_{\Gamma_m^\varepsilon} = \kappa [M_\varepsilon]_{\Gamma_m^\varepsilon} & \text{on } \Gamma_m^\varepsilon \times]0, T_E[\\ [\hat{\mathcal{D}}_{0\varepsilon} \nabla M_\varepsilon \cdot \nu]_{\Gamma_m^\varepsilon} = 0 & \text{on } \Gamma_m^\varepsilon \times]0, T_E[\\ M_\varepsilon(\cdot, 0) = M_{ini} & \Omega^\varepsilon \end{cases}$$

Transformed equation





Scaled periodic cell Y,
Zoom $\sim 1/\varepsilon$

Change of variables: $\widetilde{M}_\varepsilon(\mathbf{x},t) = M_\varepsilon(\mathbf{x},t)e^{\iota q \mathbf{u_q}\cdot\mathbf{x}\int_0^t f(s)\,ds}$.

$$
\begin{cases}
\dfrac{\partial}{\partial t}\widetilde{M}_\varepsilon(\mathbf{x},t) - \operatorname{div}\left(\mathcal{D}_{0\varepsilon}(\mathbf{x})\nabla\widetilde{M}_\varepsilon(\mathbf{x},t) - \iota q\mathbf{u_g}F(t)\mathcal{D}_{0\varepsilon}(\mathbf{x})\widetilde{M}_\varepsilon(\mathbf{x},t)\right) & \\
\quad + \iota q\mathbf{u_g}F(t)\mathcal{D}_{0\varepsilon}(\mathbf{x})\nabla\widetilde{M}_\varepsilon(\mathbf{x},t) + q^2F(t)^2\mathcal{D}_{0\varepsilon}(\mathbf{x})\widetilde{M}_\varepsilon(\mathbf{x},t) = 0 & \text{in } \Omega^\varepsilon\times]0,T_E[\\
\mathcal{D}_{0\varepsilon}\nabla\widetilde{M}_\varepsilon\cdot\nu - \iota q\mathbf{u_g}F(t)\mathcal{D}_{0\varepsilon}\widetilde{M}_\varepsilon\cdot\nu = \kappa^\varepsilon[\widetilde{M}_\varepsilon]_{\Gamma_m^\varepsilon} & \text{on } \Gamma_m^\varepsilon\times]0,T_E[\\
[\mathcal{D}_{0\varepsilon}\nabla\widetilde{M}_\varepsilon\cdot\nu - \iota q\mathbf{u_g}F(t)\mathcal{D}_{0\varepsilon}\widetilde{M}_\varepsilon\cdot\nu]_{\Gamma_m^\varepsilon} = 0 & \text{on } \Gamma_m^\varepsilon\times]0,T_E[\\
\widetilde{M}_\varepsilon(\cdot,0) = M_{init} & \text{in } \Omega^\varepsilon
\end{cases}
$$

Assignments

Fix as time profile the one associated to the PGSE and

1. Solve the transformed BT-equation inside 2-compartment geometry composed by a periodic square with a circle inside using FreeFeem++
2. Solve the transformed BT-equation only inside a Circle using FreeFeem++

Hint:
Use the “matrix version” of the Theta method

$$u_{k+1} = u_k + h[\theta f(x_{k+1}, u_{k+1}) + (1-\theta) f(x_k, u_k)]$$

Create the Geometry

```
load "lapack"
// Solve Bloch Torrey
//
verbosity  = 0;
//
real dim = 2; // setting dimension=2 <-- R^2
//
// parameter for the problem.
real epsilon = 1.0;
real kappa = 5.0e-5;
real sigmaE = 3e-3, sigmaC = 1.6e-3; // setting piece-wise constant diffusion coefficient

// Defining parameter to create the domain.
real R = 0.3*epsilon; // radius of the circle.
real xstart = -epsilon/2, xend = epsilon/2, ystart = -epsilon/2, yend = epsilon/2; // limits of the square.
real square1 = 1, square2 = 2, square3 = 3, square4 = 4, circle = 5; // label of the border
//

// Defining the square.
border Omega1(xi=0,1){x = xend-(xend-xstart)*xi;      y = ystart;             label=square1;}
border Omega2(xi=0,1){x = xstart;             y = ystart+(yend-ystart)*xi;   label=square2;}
border Omega3(xi=0,1){x = xstart+(xend-xstart)*xi;    y = yend;               label=square3;}
border Omega4(xi=0,1){x = xend;               y = yend-(yend-ystart)*xi;     label=square4;}
// Defining the internal circle.
border Gamma(xi=0,2*pi){x=R*cos(xi);    y=R*sin(xi);    label=circle;}
// plot the border
plot(Omega1(-120) + Omega2(-120) + Omega3(-120) +Omega4(-120) + Gamma(-300), wait=true);//, ps="bord.eps");
```

Create the mesh and set the finite elements

```
29  //
30  //
31  // Creating and plot the mesh inside the circle and outside the circle.
32  // extracellular (outside the circle):
33  int Na = 120;
34  int Nb = 300;
35  mesh Ye = buildmesh(Omega1(-Na) + Omega2(-Na)+ Omega3(-Na) + Omega4(-Na) + Gamma(-Nb));
36  // intracellular (inside the circle):
37  mesh Yc = buildmesh(Gamma(Nb));
38  // plot mesh
39  plot(Yc, wait=true);//, ps="Yc.eps"); // intracellular
40  plot(Ye, wait=true);//, ps="Ye.eps"); // extracellular
41  //
42  //-----------------------------------------------------------------------------
43  //
44  // Define finite elements spaces.
45  fespace Yehp(Ye,P1,periodic=[[square2,y],[square4,y],[square1,x],[square3,x]]);
46  fespace Ych(Yc,P1);
47  Yehp<complex> Me, Ve, Meinit;
48  Yehp Mereal;
49  Ych<complex> Mc, Vc, Mcinit;
50  Ych Mcreal;
51  //
52  // Compute |Ye|, |Yc|, |Y| and |Gamma|
53  real measYe = int2d(Ye)(1.0);
54  real measYc = int2d(Yc)(1.0);
55  real measY = measYe + measYc;
56  real measGamma = int1d(Ye, circle)(1.0);
57  //
58  //-----------------------------------------------------------------------------
```

Defining the PGSE

```
58  //-----------------------------------------------------------------------
59  //
60  // parameter for PGSE
61  real delta = 3.5e+3;    // space interval for the time profile PGSE
62  real tdiff = 7.0e+3;    // diffusion time
63  real t1 = 0.0;          // initial time of PGSE
64  real tend = t1 + delta + tdiff; // TE
65  // Defining PGSE
66  func real F(real t)
67  {return 0.0 + (t-t1)*(t1<t)*(t<=t1+delta)
68              + delta*(t1+delta<t)*(t<=t1+tdiff)
69              + (delta-(t-(t1+tdiff)))*(t1+tdiff<t)*(t<=t1+tdiff+delta);}
70  //
71  // intensity and direction of the gradient.
72  real bval = q^2*delta^2*(tdiff-delta/3.0);
73  real q = sqrt(bval/(delta^2*(tdiff-delta/3)));
74  real[int] normal = [1.0,0.0];
75  real nu1 = normal(0), nu2 = normal(1);
76
```

Parameters for the theta-method

```
78  // Parameter for Theta-method
79  real deltatest = 2.5e+3;
80  real tdifftest = 2.5e+3;
81  real btest = 50;
82  real qtest = sqrt(btest/(deltatest^2*(tdifftest-deltatest/3)));
83  real dttest = 88;
84  real tinit = 0.0;
85  real t;
86  real dt = min(dttest*qtest/q, delta/10.0);//(tend - tinit)/Ntstep; // time step
87  real Ntstep = (tend-tinit)/dt;//2000; // number of time step
88  cout << "dt = " << dt << endl;
89  //
```

Variational form

```
// Defining the variational form for BT-equation to compute the stiff matrix and assemble them.
varf varKee(Me, Ve) =
        int2d(Ye)(sigmaE*(dx(Me)*dx(Ve)+dy(Me)*dy(Ve)))
        +int1d(Ye,circle)(kappa*Me*Ve);
//
varf varKec(Me, Vc)=
        -int1d(Ye,circle)(kappa*Me*Vc);
//
varf varKce(Mc, Ve)=
        -int1d(Yc,circle)(kappa*Mc*Ve);
//
varf varKcc(Mc, Vc) =
        int2d(Yc)(sigmaC*(dx(Mc)*dx(Vc)+dy(Mc)*dy(Vc)))
        +int1d(Yc,circle)(kappa*Mc*Vc);
//
varf varKeet(Me,Ve) =
        -int2d(Ye)(1i*q*sigmaE*(nu1*dx(Ve)+nu2*dy(Ve))*Me)
        +int2d(Ye)(1i*q*sigmaE*(nu1*dx(Me)+nu2*dy(Me))*Ve);
//
varf varKcct(Mc,Vc) =
        -int2d(Yc)(1i*q*sigmaC*(nu1*dx(Vc)+nu2*dy(Vc))*Mc)
        +int2d(Yc)(1i*q*sigmaC*(nu1*dx(Mc)+nu2*dy(Mc))*Vc);
//
varf varKeett(Me,Ve) =
        int2d(Ye)(q^2*sigmaE*Me*Ve);
//
varf varKcctt(Mc,Vc) =
        int2d(Yc)(q^2*sigmaC*Mc*Vc);
//
varf varMassE(Me, Ve) =
        int2d(Ye)(Me*Ve/dt);
//
varf varMassC(Mc, Vc) =
        int2d(Yc)(Mc*Vc/dt);
//
```

Compute the matrices and set the parameters

```
125  // Compute the stiff matrix.
126  matrix <complex> Kee = varKee(Yehp, Yehp);
127  matrix <complex> Kcc = varKcc(Ych, Ych);
128  matrix <complex> Kec = varKec(Yehp, Ych);
129  matrix <complex> Kce = varKce(Ych, Yehp);
130  //
131  matrix <complex> Keet = varKeet(Yehp, Yehp);
132  matrix <complex> Kcct = varKcct(Ych, Ych);
133  matrix <complex> Kcet=0.0*Kce, Kect=0.0*Kec;
134  //
135  matrix <complex> Keett = varKeett(Yehp, Yehp);
136  matrix <complex> Kcctt = varKcctt(Ych, Ych);
137  //
138  matrix <complex> K = [[Kee, Kce],
139                        [Kec, Kcc]];
140  matrix <complex> Kt = [[Keet, Kcet],
141                         [Kect, Kcct]];
142  matrix <complex> Ktt = [[Keett, Kcet],
143                          [Kect, Kcctt]];
144  //
145  // Compute mass matrix
146  matrix <complex> MassE = varMassE(Yehp, Yehp);
147  matrix <complex> MassC = varMassC(Ych, Ych);
148  matrix <complex> Massce=Kcet, Massec=Kect;
149  //
150  matrix <complex> Mass = [[MassE, Massce],
151                           [Massec, MassC]];
152  matrix <complex> A, B;
153  real theta = -0.5;
154  cout << theta << endl;
155  //
156  //
157  complex[int] M(K.n), Mold(K.n);//, Mcheck(Kcc.n), MoldCheck(Kcc.n);
158  //cout << "M = " << M << endl;
159  Meinit = 1.0; Mcinit = 1.0; // <------- initial condition.
160  M=[Meinit[],Mcinit[]];
```

Implementing the Theta-method

```
163  t = tinit; // <-------- starting time
164  //
165  // Solving with theta-method
166      for(int step=1; step<=Ntstep; step++) // <-- Starting Euler
167      {
168          // Compute PGSE
169          F(t); F(t+dt);
170          // Update matrix B
171
172          B = Mass + theta*F(t)*Kt + theta*(F(t)^2)*Ktt + theta*K;
173          A = Mass + (1.0+theta)*K + (1.0+theta)*F(t+dt)*Kt + (1.0+theta)*(F(t+dt)^2)*Ktt;
174          set(A,solver=UMFPACK);
175          //
176          // Update solution
177          Mold = B*M; //Mold = -Mold;
178          M = A^-1*Mold;
179          // Update time.
180          t+=dt;
181      }
182  // Solution of BT-equation.
183  [Me[],Mc[]] = M;
184  // plot the final magnetization
185  Mereal = real(Me);
186  Mcreal = real(Mc);
187  plot(Mereal, value=1, fill=1, wait=true);//, ps="exterior.eps");
188  plot(Mcreal, value=1, fill=1, wait=true);//, ps="interior.eps");
189  //
```