The `swimmer` Java library in `clasJLib`

The `swimmer` library in the clas12 repository has been refactored. It wasn't ready for prime time—and I don't think anyone was using it. But if so, please note that the API has changed, as well as the input units.

The input units are now:

- Meters for any lengths
- Degrees for any angles
- GeV (modulo factors of *c*) for mass, energy and momentum

We anticipate that any modifications to `swimmer` will be backwards compatible from this point forward.

The `swimmer` library depends on the `magfield` library (also in the repository and in also in `clasJLib`). The `magfield` library is standalone.

The `swimmer` library will integrate particles through a magnetic field. It is not CLAS specific—but at the moment the `magfield` library, which it uses to obtain field vectors, only has map file for the CLAS 12 torus and solenoid.

The `swimmer` library uses the Lund particle designations. Support for the Lund format is provided in the `cnuphys.lund` package within the `swimmer` library. We probably should consider making a common, standalone lund Java library.

As of 12/1/13, the swimmer library was based on a constant-step Runge-Kutta 4 integrator. We are working on an adaptive stepsize RK4 for the next release.

The basic steps to swim a particle are:

1. Use magfield to create an **IField** object from one or more field maps.
2. Use the **IField** object to create a **Swimmer** object. The **Swimmer** can be used to swim any number of particles, and it is thread safe. You only need a new **Swimmer** if you change magnetic fields.
3. Obtain a **LundId** object for the particle you want to swim.
4. Set the initial and stopping conditions.
5. Swim the particle.

The swimming has two basic modes of operation. One mode, the *trajectory* mode, is when the swimming as carried out and the results are returned in a **SwimTrajectory** object. This is useful if you want to store the trajectory at a certain number of points. For example, *ced* uses this mode so that after the particle is swum *ced* can redraw the trajectory as needed without reswimming. The other mode of operation is the *listener* mode. In the listener mode, a listener is called at every advance of the integration, but no trajectory is cached and no **SwimTrajectory** object is returned.

Examples

Note: there is a class `cnuphys.swim.Example` in the simmer library where you can find the examples here coded and tested.

Reading the magnetic fields

The **Example** class uses a method `getMagneticFields()` to load the torus and solenoid. It works on the assumption that the current working directory is at the same level as clasJLib, where the field maps are stored in the data folder.

```
	private static Torus torus;
	private static Solenoid solenoid;
	private static CompositeField compositeField;

	//tries to get the magnetic field assuming it is in clasJLib
	private static void getMagneticFields() {
		//will read mag field assuming we are in a
		//location relative to clasJLib. This will
		//have to be modified as appropriate.

		String clasJLib = "../clasJLib";
		//see if it is a good location
		File file = new File(clasJLib);
		if (!file.exists()) {
			System.err.println("dir: " + clasJLib + " does not exist.");
			System.exit(1);
		}

		//OK, see if we can create a Torus
		String torusFileName = clasJLib +
"/data/torus/v1.0/clas12_torus_fieldmap_binary.dat";
		File torusFile = new File(torusFileName);
		try {
			torus = Torus.fromBinaryFile(torusFile);
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}

		//OK, see if we can create a Solenoid
		String solenoidFileName = clasJLib + "/data/solenoid/v1.0/solenoid-srr.dat";
		File solenoidFile = new File(solenoidFileName);
		try {
			solenoid = Solenoid.fromBinaryFile(solenoidFile);
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}

		//OK, see if we can create a composite field
		compositeField = new CompositeField();

		//print some features
		if (torus != null) {
			compositeField.add(torus);
		}
		if (solenoid != null) {
			compositeField.add(solenoid);
		}

		//change sign of torus field so electrons bend toward beamline
		torus.setInvertField(true);

	}
```

Common Setup

There is some common setup regardless of what integration mode we choose. Suppose we want to integrate an electron with kinetic energy of 1 GeV from the nominal target position (0, 0, 0) with a theta of 30° and a phi of 0. And suppose we only want to use the torus.

```
//create a swimmer for our magnetic field
Swimmer swimmer = new Swimmer(torus);

//OK lets integrate an electron and see what we get
LundId electron = LundSupport.getInstance().get(11);

//vertex position
double xo = 0.0;
double yo = 0.0;
double zo = 0.0;

//initial angles in degrees
double theta = 30.0;
double phi = 0.0;

//these will be used to create a DefaultStopper
double rmax = 7.0; //m
double maxPathLength = 8.0; //m

//step size in m
double stepSize = 5e-4; //m

//The  momentum, if the KE = 1 GeV
double momentum = electron.pFromT(1.0);
```

Trajectory Mode

In the trajectory mode we want to save all the steps. We do that here and then generate a crude console plot to see if we get something reasonable.

```
//save about every 20th step
        double distanceBetweenSaves = 20*stepSize;

//swim the particle, and return the results in a SwimTrajectory object
        SwimTrajectory traj = swimmer.swim(electron, xo, yo, zo, momentum, theta, phi,
                        rmax, maxPathLength, stepSize, distanceBetweenSaves);

        //how many steps did we save:
        System.out.println("Trajectory has: " + traj.size() + " stored points");


        //lets create a crude terminal plot
        double xx[] = new double[traj.size()];
        double zz[] = new double[traj.size()];
        int index = 0;
        for (double v[] : traj) {
                xx[index] = 100*v[0]; //convert to cm
                zz[index] = 100*v[2];
                index++;
        }
        TerminalPlot.plot2D(80, 20, "z (horizontal, cm) vs. x (vertical, cm)", zz, xx);
```

The “plot” matches what *ced* produces:

```
Trajectory has: 706 stored points

                              z (horizontal, cm) vs. x (vertical, cm)

2.63e+02  |+---------+---------+---------+---------+---------+---------+---------+---------|
          |                                                                               *|
          |                                                                       ******** |
          |                                                                ********        |
          |                                                         ********               |
          |                                                  ********                      |
          |                                             ******                             |
          |                                        *****                                   |
          |                                    *****                                       |
          |                                *****                                           |
1.38e+02  +                             ****                                               +
          |                          ****                                                  |
          |                       ****                                                     |
          |                    ****                                                        |
          |                 ****                                                           |
          |              ****                                                              |
          |           ****                                                                 |
          |        ****                                                                    |
          |     ****                                                                       |
          |  ****                                                                          |
0.00e+00  +***                                                                             +
          |+---------+---------+---------+---------+---------+---------+---------+---------|
       0.00e+00  8.10e+01  1.62e+02  2.43e+02  3.24e+02  4.05e+02  4.86e+02  5.67e+02  6.48e+02
```

Listener Mode

We use this mode when we are not interested in a trajectory. It will call a listener at every integration step. He we use a **DefaultListener** as an example. It simply caches the last step it is given and the total number of steps.

```
//same problem using a listener and a default stopper

               DefaultListener listener = new DefaultListener();
               DefaultSwimStopper stopper = new DefaultSwimStopper(xo, yo, zo, rmax,
maxPathLength);

               swimmer.swim(electron, xo, yo, zo, momentum,
                              theta, phi, stopper, listener,
                              maxPathLength, stepSize);

               double lastY[] = listener.getLastPosition();
               System.out.println("\nresult from listener method");
               System.out.println(String.format("count = %d  t = %7.4e v = [%7.4f, %7.4f,
%7.4f]",
                              listener.getCount(),
                              listener.getLastTime(),
                              lastY[0], lastY[1], lastY[2]));
```

This produces the output:

```
result from listener method
count = 14117  t = 2.3545e-08 v = [ 2.6310,  0.0000,  6.4869]
```