Export to '.tmat.hdf5' format with Julia

baptiste

This document describes the process of exporting T-matrices in the standardised HDF5 format in Julia. For illustration, we start by producing a dummy dataset.

Mockup input data

For convenience, we store compound objects as `Dict`tionaries, which will then be mapped into HDF5 groups during export.

```
using Pkg, UUIDs

# possibly multiple wavelengths
wavelength  = collect(400:50:800)
Nl = length(wavelength)

# dummy 30x30 matrix values for each wavelength
tmatrix = reshape(repeat(collect(1:900.0) + collect(1:900)*1im, Nl),
                  (30,30,Nl))

modes = Dict("l" => collect(1:30),
             "m" => collect(1:30),
             "polarisation" => repeat(["electric","magnetic"],  15))

# dummy 'analytical zeros' for e.g. EBCM methods
azeros = collect(Iterators.product(1:2:30, 1:2:30))
analytical_zeros = Dict("p" => [z[1] for z in azeros[:]],
                        "pp" => [z[2] for z in azeros[:]])

materials = Dict("embedding" => Dict("relative_permeability" => 1.0,
                                     "relative_permittivity" => 1.33^2),
                 "Au" => Dict("relative_permeability" => 1.0,
                              "relative_permittivity" => -11.4+1.181im))
```

```
geometry = Dict("shape" => "spheroid",
                "radiusxy" => 20.0,
                "radiusz" => 40.0)

# details about computation, including full script
computation = Dict("method" =>"EBCM",
                   "software" => "SMARTIES",
                   "version" => "1.1",
                   "Ntheta" => 40,
                   "accuracy" => 1e-10,
                   "analytical_zeros" => analytical_zeros,
                   "script" => read("test_dummy.jl", String))

uuid = string(UUIDs.uuid4())

pkgs = Pkg.Operations.Context().env.manifest
hdf5version = string(pkgs[findfirst(v->v.name == "HDF5", pkgs)].version)
```

Saving to HDF5

The `HDF5.jl` library provides all the tools required, and a relatively high-level interface. Unlike R and Matlab, there does not seem to be a built-in high-level way to store `Dict` objects, so I wrote the following wrapper.

```
function write_dict(groupname, dict::Dict)
    g = create_group(fid, groupname)
    for (key, value) in dict
        keyname = groupname * "/" * key
        if typeof(value) <: Dict # nested, recurse
            write_dict(keyname, value)
        else
            write(fid, keyname, value)
        end
    end
end
```

Complex arrays are stored with `r` and `i` fields as usual. Note that Julia has column-major ordering, unlike HDF5, and we therefore explicitly transpose the matrix before saving it, to avoid confusion if the data are to be read in another language. I did not find how to link the embedding medium from materials to the top level, so it is simply duplicated.

```
using HDF5

f = "aj.tmat.h5"
fid = h5open(f, "w")

# write top-level datasets
fid["vacuum_wavelength"] = wavelength
fid["tmatrix"] = tmatrix
fid["uuid"] = uuid

# write groups (some nested)
write_dict("embedding", embedding)
write_dict("materials", materials)
write_dict("geometry", geometry)
write_dict("modes", modes)
write_dict("computation", computation)

# write attributes
attributes(fid)["name"] = "Au prolate spheroid in water"
attributes(fid)["created_with"] = "HDF5.jl"
attributes(fid)["storage_format_version"] = hdf5version
attributes(fid)["description"] = "Computation using SMARTIES, a numerically robust EBCM imple
attributes(fid)["keywords"] = "gold, spheroid, ebcm"
attributes(fid["vacuum_wavelength"])["unit"] = "nm"
attributes(fid["uuid"])["version"] = "4"
attributes(fid["geometry"])["name"] = "prolate spheroid"

close(fid)
```

```
% save to file
f = 'am.tmat.h5';
saveh5(a, f, 'ComplexFormat', {'r','i'}, 'rootname', '', ...
       'Compression', 'deflate'); % compression is optional

% write polarization separately
h5create(f,'/modes/polarization', size(polarization), 'Datatype', 'string')
h5write(f,'/modes/polarization', polarization)
```

Attributes are written in a separate step.

```
% write root attributes
h5writeatt(f, '/', 'name', 'Au prolate spheroid in water');
h5writeatt(f, '/', 'created_with', 'Matlab easyh5');
h5writeatt(f, '/','keywords', 'gold, spheroid, ebcm');
[maj,min,rel] = H5.get_libversion();
h5writeatt(f, '/', 'storage_format_version', sprintf('%d.%d.%d',maj,min,rel));
h5writeatt(f, '/','description', ...
    'Computation using SMARTIES, a robust EBCM for spheroids');

% attributes of specific objects
h5writeatt(f, '/vacuum_wavelength', 'unit', 'nm');
h5writeatt(f, '/uuid', 'version', '4');
h5writeatt(f, '/geometry', 'name', 'prolate spheroid');
```