PostgreSQL access with Matlab

For accessing data from PostgreSQL with geospatial encoding via PostGIS, we use the `postgresql` toolbox from OpenEarthTols. This toolbox uses the native matlab database toolbox license, but also works without this license, by using the JDBC4 java driver directly. You can obtain the OpenEarthTools mallab toolbox after free registration for the Deltares open source software (oss) portal, where you can also download Delft3D.

Load the java JBDC driver. You need to do this only once per matlab session. We do not load this by default when running `oetsettings`, to limit mremeory use of you not need it. Loading the JBDC driver is required both when you have a matlab database toolbox license, and when you have not.

```
pg_settings
```

Now connect to a database by specifying the access credentials. Note we provide free test user credentials here, which can sometimes fail when too many users are logged in, you will get this error
message in that case: *??? Java exception occurred: org.postgresql.util.PSQLException: FATAL: sorry, too many clients already*

```
OPT.db               = 'ICES';
OPT.user             = 'dbices';
OPT.pass             = 'vectors';
OPT.host             = 'postgresx03.infra.xtr.deltares.nl';
OPT.database_toolbox = 0; % disallow native matlab database toolbox, and force direct ofse of JDBC4 java library

conn=pg_connectdb(OPT.db,'user',OPT.user,'pass',OPT.pass,...
                         'database_toolbox',OPT.database_toolbox,'host',OPT.host);
```

Now view the contents of the database

```
pg_dump(conn)
```

THis will generate an ASCII table that looks like a netCDF dump.

```
    ocean (12920408):
    sdepth_bin (character varying)
    ices_area (character varying)
    statsq (character varying)
    the_point (USER-DEFINED)
    ...
    south (double precision)
    icesname (character varying)
    id (double precision)
    gid (integer) [PK]
```

Now prepare SQL statements and execute to obtain the data in the matlab workspace

```
strSQL = 'select the_point, st_x(st_transform(the_point,28992)), st_y(st_transform(the_point,28992)) from ocean
limit 10'
D = pg_fetch(conn,strSQL);
```

which yields

```
[297.6903]
```

```
strSQL = 'select avg(doxy) from ocean o, icessquares i where st_contains(i.the_geom, o.the_point) and year =
2005 and i.statsq = ''32F3'''
D = pg_fetch(conn,strSQL)
```

which yields

```
    [1x50 char]    [ 6.9346e+005]    [7.8723e+005]
    [1x50 char]    [ 6.9346e+005]    [7.8723e+005]
    [1x50 char]    [ 6.9346e+005]    [7.8723e+005]
    [1x50 char]    [ 6.9346e+005]    [7.8723e+005]
    [1x50 char]    [ 6.9346e+005]    [7.8723e+005]
    [1x50 char]    [ 6.9346e+005]    [7.8723e+005]
    [1x50 char]    [ 6.9346e+005]    [7.8723e+005]
    [1x50 char]    [-2.3881e+005]    [2.7697e+006]
    [1x50 char]    [ 6.9346e+005]    [7.8723e+005]
    [1x50 char]    [ 6.9346e+005]    [7.8723e+005]
```

The first column is a hexadecimal PostGIS object. You can unwrap this with the JTS toolkit in matlab.
This will show that this data is native encoded in WGS84

```
SRID=4326;POINT(13.7667 54.7833)
```

And finally, CLOSE YOUR DATABASE CONNECTION when you are done.

```
conn.close
```

Any query after this, will result in an error:
??? Java exception occurred: org.postgresql.util.PSQLException: This connection has been closed.

See also: PostgreSQL access with python, PostgreSQL access with R