Please, insert
logo here :`(

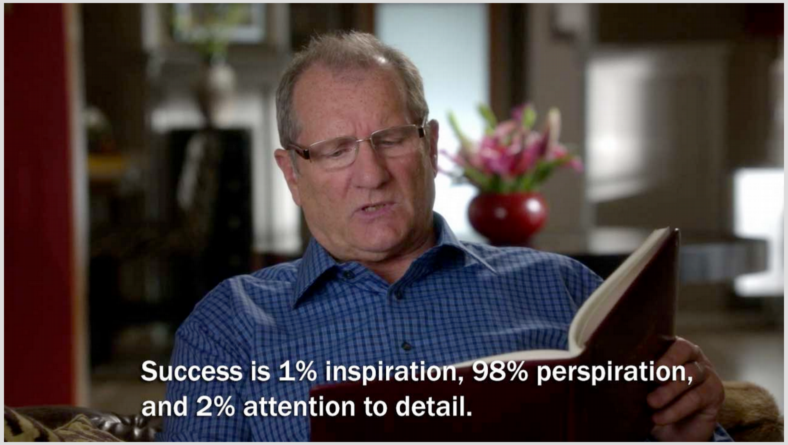
GNSS Field Analysis (GFA)

Vicente Javier Yáñez Cuadra

URL: https://github.com/VicenteYanez/GFA

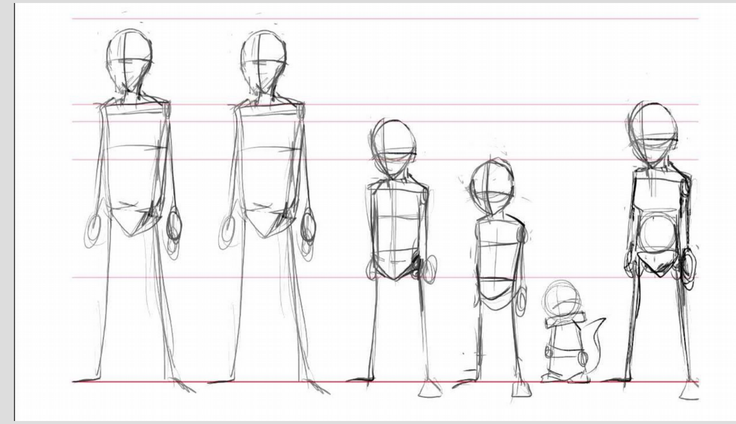
Alpha version presentation

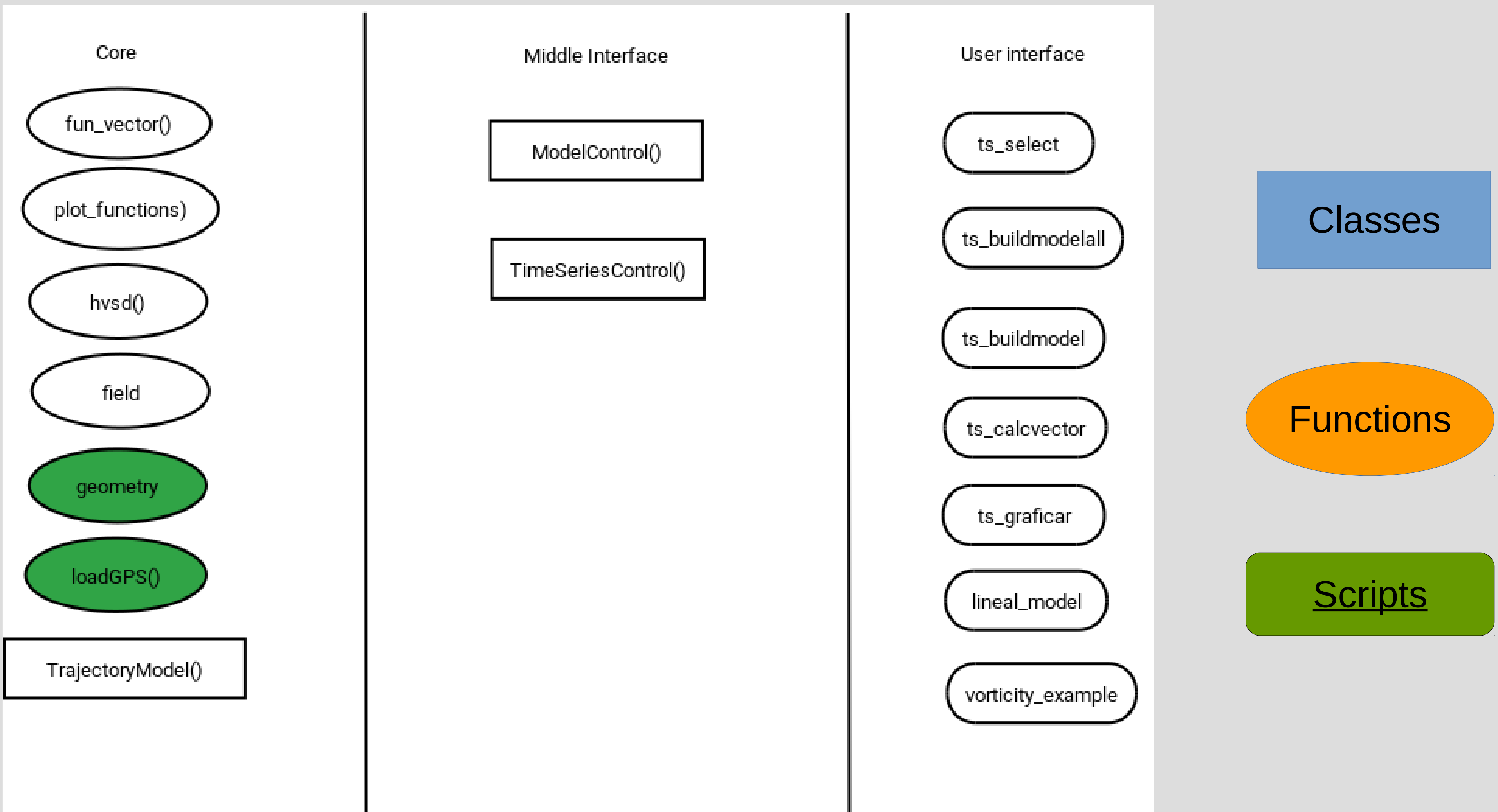
Project Philosophy



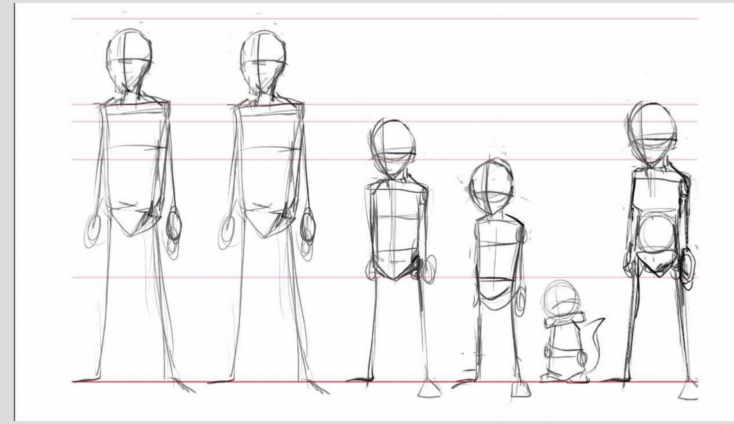
- GFA is free software and bla bla bla…
- GFA main goal is group all the necessary functions for the analysis of the GNSS time series data in geoscience.
- The project have two set of code: the GNSS analysis code and the field analysis code.
- Plus, the project have other three different sets of code: the core functional code, the middle interface and the user-interface(ui) code.
  - The core functional code shouldn't be change (to maintain the compatibility with the ui).

Basic Structure



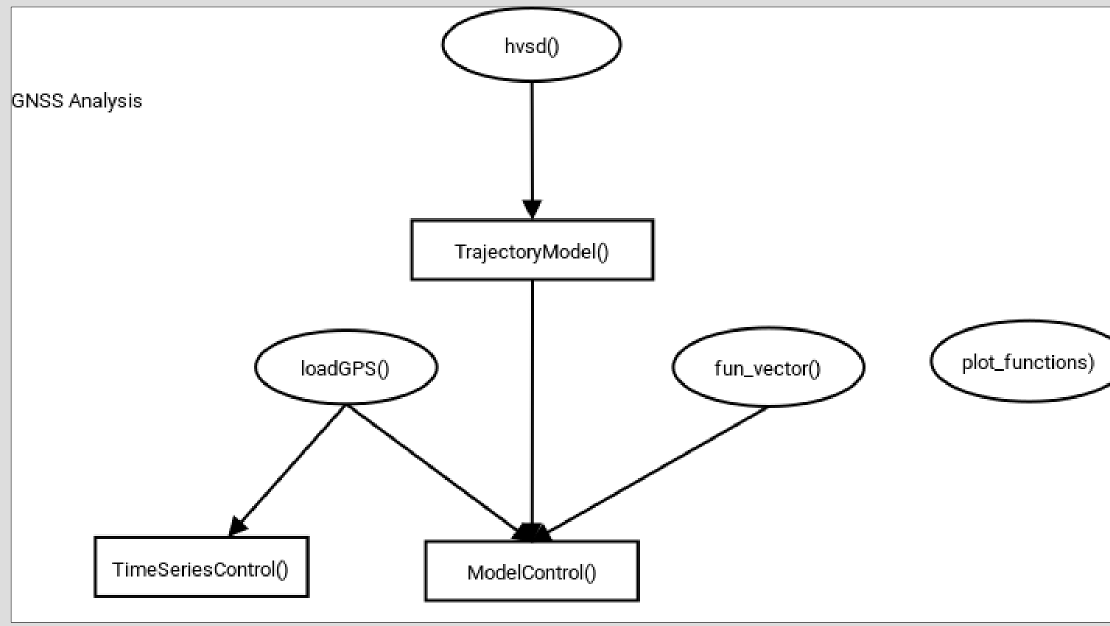


Basic Structure



GFA is composed of two parts

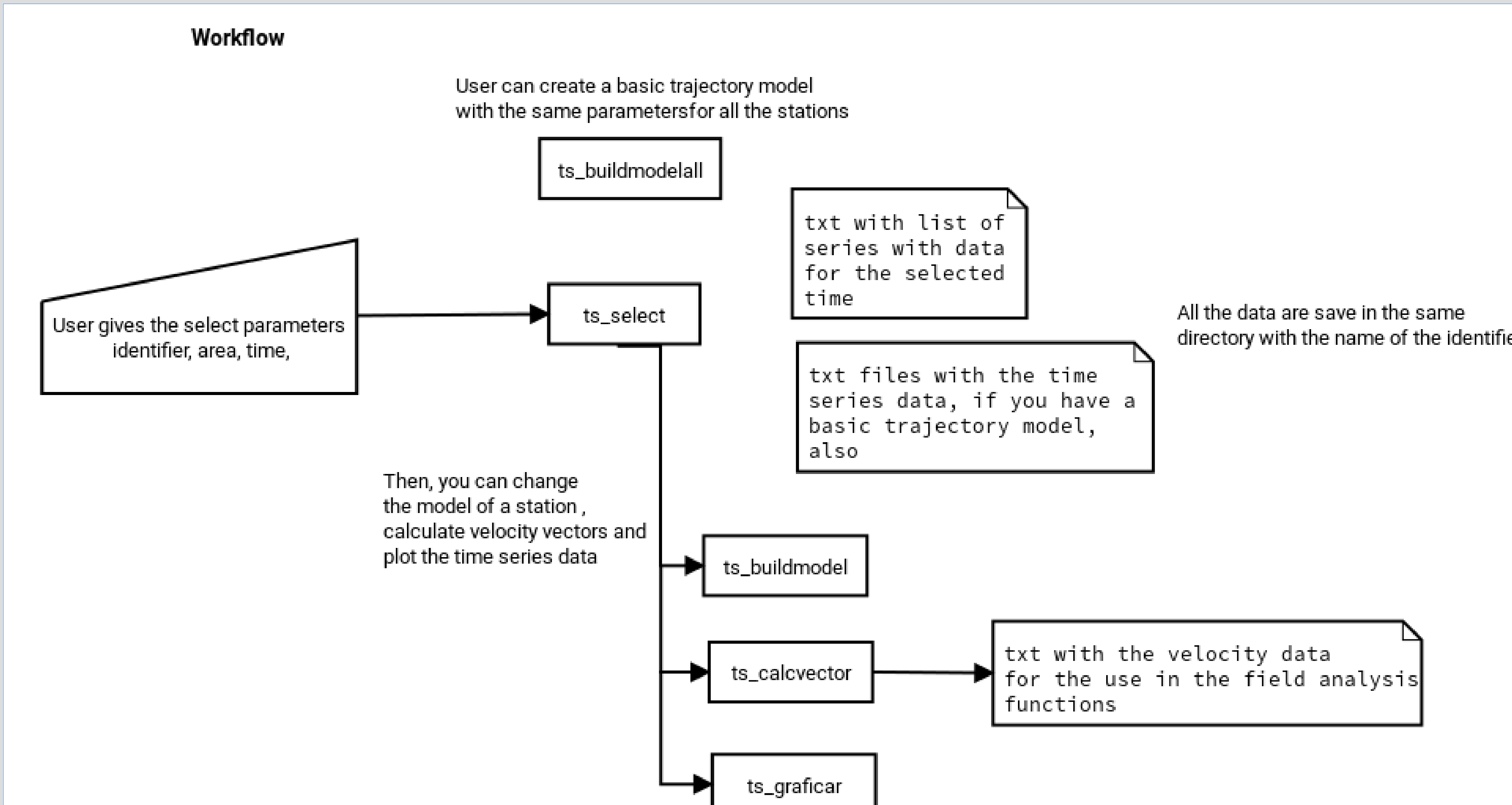
- **GNSS Analysis**
  - This have many classes, scripts and functions.
- **Field Analysis**
  - Only have two function files.
    - field.py
    - geometry.py



Functions

Classes

Predefined Scripts Workflow



GNSS Analysis: TrajectoryModel()

$$\mathbf{x}(t) = \sum_{i=1}^{n_P+1} \mathbf{p}_i (t - t_R)^{i-1} + \sum_{j=1}^{n_J} \mathbf{b}_j H(t - t_j) + \sum_{k=1}^{n_F} \mathbf{s}_k \sin(\omega_k t) + \mathbf{c}_k \cos(\omega_k t) + \sum_{i=1}^{n_T} \mathbf{a}_i \log(1 + \Delta t_i / T_i)$$

- How to use TrajectoryModel()

```
modelo = ModeloTrayectoria(tiempo)

modelo.n = param['polinomio']

modelo.tjump = np.array(param['saltos'])

modelo.fperiods = np.array(param['Periodos
Fourier'])

modelo.tlt = np.array(param['Inicio log'])

modelo.tsc = np.array(param['Escala curva log'])


mod_e, res_e =
model.modelo_trayectoria(desplaz_e)
```

Principal Method

- modelo_trayectoria, that calls to trend(), jump(), cycle() and lgt() methods.
- Each method create a specific part of the equation

Other methods

- save_components()

GNSS Analysis: How the function that calculate the trajectory model works?

$$\mathbf{x}(t) = \sum_{i=1}^{n_P+1} \mathbf{p}_i (t - t_R)^{i-1} + \sum_{j=1}^{n_J} \mathbf{b}_j H(t - t_j) + \sum_{k=1}^{n_F} \mathbf{s}_k \sin(\omega_k t) + \mathbf{c}_k \cos(\omega_k t) + \sum_{i=1}^{n_T} \mathbf{a}_i \log(1 + \Delta t_i / T_i)$$

The code...

```
# solucion mediante minimos cuadrados
parametros, residual, rank, s = np.linalg.lstsq(modelo, y)
# devuelve el valor de distancia del modelo para cada componente
res_nt = np.matrix(modelo[:, :nt])*np.matrix(parametros[:nt]).T
res_nj = np.matrix(modelo[:, nt:n2])*np.matrix(parametros[nt:n2]).T
res_nf = np.matrix(modelo[:, n2:n3])*np.matrix(parametros[n2:n3]).T
res_nl = np.matrix(modelo[:, n3:n4])*np.matrix(parametros[n3:n4]).T
```

Mathematically…

(t is an array of length *m*)

$$\begin{bmatrix} p_1 \\ p_2 \\ \dots \\ p_{np} \\ b_1 \\ b_2 \\ \dots \\ b_{nj} \\ \dots \\ a_{nl} \end{bmatrix} = lstq(\begin{bmatrix} (t-t_R)^0 \\ (t-t_R)^1 \\ \dots \\ (t-t_R)^{np} \\ (t-t_1) \\ (t-t_2) \\ \dots \\ (t-t_{nj}) \\ \dots \\ log(1+\nabla t_n/T_{nl}) \end{bmatrix}, \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_m \end{bmatrix})$$

$$trend_{m\times 1} = \left[(t-t_R)^0 \quad (t-t_R)^1 \quad \dots \quad (t-t_R)^n\right]_{m\times n} \times \begin{bmatrix} p_1 \\ p_2 \\ \dots \\ p_n \end{bmatrix}_{1\times n}$$

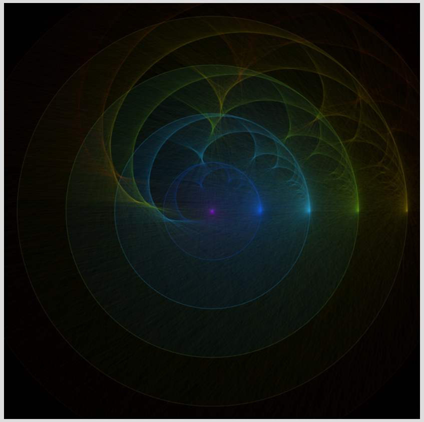
GNSS Analysis: TimeSeriesControl() & ModelControl()

- Principal classes that work with massive time-series data. Their mission is create a middle interface, between the pure functional code and the scripts created by the user.
- Principal method for the two classes is load_estation()/load_model, which returns the time-series/model data
- ModelControl() also have the methods to create and modify a model.

```
data[0] : ['name estation', [0, ..., 6]]
          Where 0 to 6:
               0. time
               1. East displacement
               2. North displacement
               3. Vertical displacement
               4. East Error
               5. North Error
               6. Vertical Error
           example:
               list[4][1][2] : north
displacement of station number 4
               list[4][0]    : name of station
number 4
```

Other functions of the ModelClass() is calc_vector() and build_model_all()

Field Analysis: field()



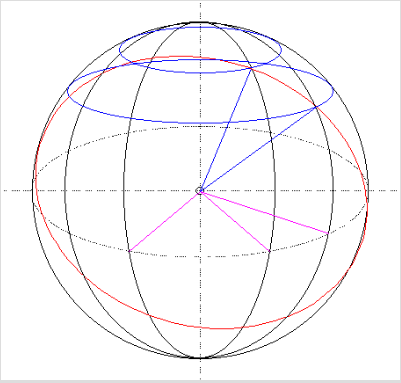
Function list:

- `triangular_gradient(ux, uy, x, y, ele)`
- `velocitytensor_2d(uxdx, uxdy, uydx, uydy)`
- `velocitytensor_3d(uxdx, uxdy, uxdz, uydx, uydy, uydz, uzdx, uzdy, uzdz)`
- `vertical_vorticity2d(tensorW)`
- `vorticity_vector3d(tensorW)`
- `kinematic_vorticity(tensorS, tensorW)`

`traza_tensor(tensor, grado)`

- `clean_array(array_pp, value_min, value_max, param_=())`

Field Analysis: geometry()

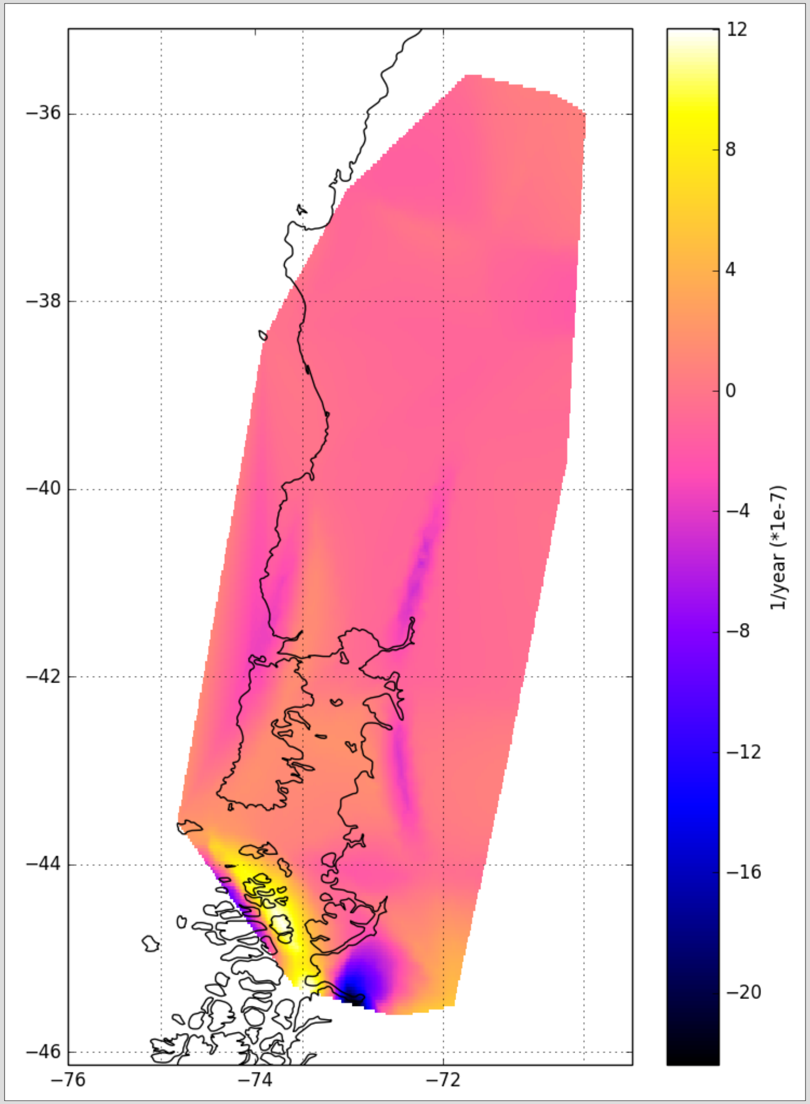


List of functions

- `rotar_sistema2d(x, y, degrees)`
- `geo2proj(x, y, x0, y0)`
- Vicenty’s functions

Field Analysis: vorticity example

```
# cargar malla
dir_path = os.path.dirname(os.path.realpath(__file__))
path_node = "{}/example_files/malla.node".format(dir_path)
path_ele = "{}/example_files/malla.ele".format(dir_path)
x_triangulos = np.loadtxt(path_node, usecols=[1], skiprows=1)
y_triangulos = np.loadtxt(path_node, usecols=[2], skiprows=1)
origen = [-76, -46]

# cargar los datos
dataset = '{}/example_files/vect2007_2010'.format(dir_path)
lon_gps = np.loadtxt(dataset, usecols=[1])
lat_gps = np.loadtxt(dataset, usecols=[2])
v_n_gps = np.loadtxt(dataset, usecols=[4])
v_e_gps = np.loadtxt(dataset, usecols=[3])

# conversion mm/año a m/año
v_e_gps = v_e_gps/1000
v_n_gps = v_n_gps/1000

# proyectar malla y puntos gps
malla_proy = geo.geo2proj(x_triangulos, y_triangulos, origen[0], origen[1])
gps_proy = geo.geo2proj(lon_gps, lat_gps, origen[0], origen[1])

# interpolar en puntos de malla triangular
vy_tri = griddata(gps_proy, v_n_gps, malla_proy, method='cubic')
vx_tri = griddata(gps_proy, v_e_gps, malla_proy, method='cubic')

# ############################################################
# Funciones field()
# ############################################################
gradiente = field.triangular_gradient(vx_tri, vy_tri, malla_proy[0],
                  malla_proy[1], path_ele)
S, W = field.velocitytensor_2d(gradiente[0], gradiente[1],
               gradiente[2], gradiente[3])
wz = field.vertical_vorticity2d(W)
# ############################################################
# Graficar #####################################################
fig = vorticity_figure(x_triangulos, y_triangulos, wz)
plt.show()
```



To do list...

- Create a configuration file to change project global variables. ***
- Create a compatibly layer with andes3db server(github branch) **
- Write the code and wiki documentation.***
- Complete the English translation.***
- Improve the quality of time series figures.*
- Package GFA like a python module.*
- Improve the interpolation method.*
- Connection with the seismic catalog.**