Cálculo de la magnitud Aproximación probabilista

Grupo de Trabajo 2 (GT2)

Workshop INNTEGRA

Método

GT1

Inputs

Longitud falla (SRL)

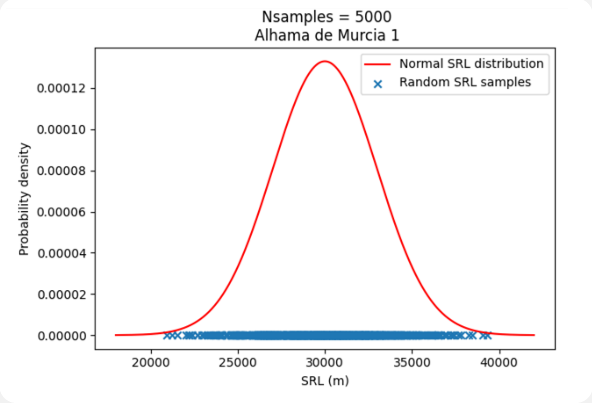
Incertidumbre (SRL_unc)

Cinemática (kinematics)

INICIO

1 Distribución de probabilidad SRL

2 Muestreo aleatorio (Monte-Carlo)



PASO 1

Muestras SRL

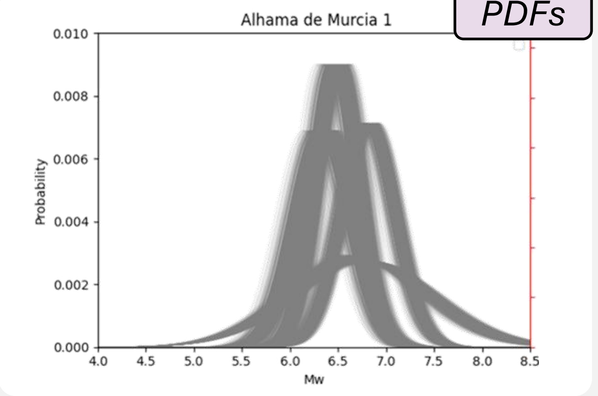
Sclaw1

Sclaw2

SclawN

Pooling de Mw + std

PDFs

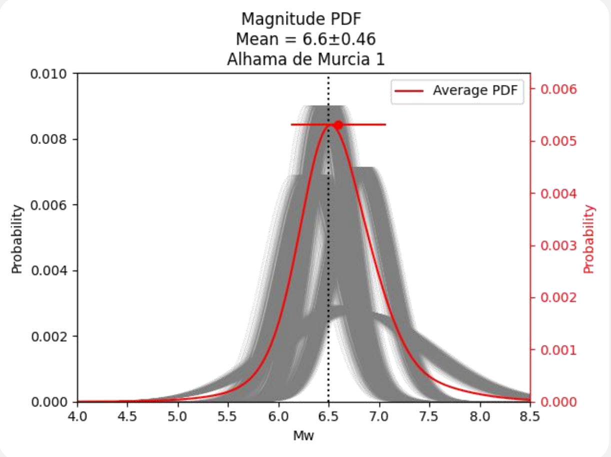


PASO 2

Método

Media de las PDFs

Distribución de magnitud final



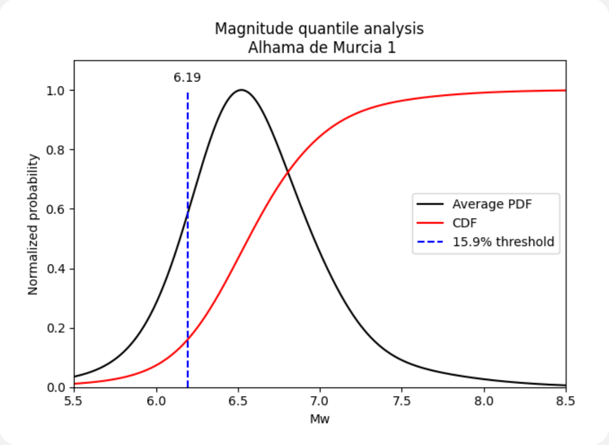
PASO 3

Umbral de Mw en el cuantil 15.9% de la CDF

SI

NO

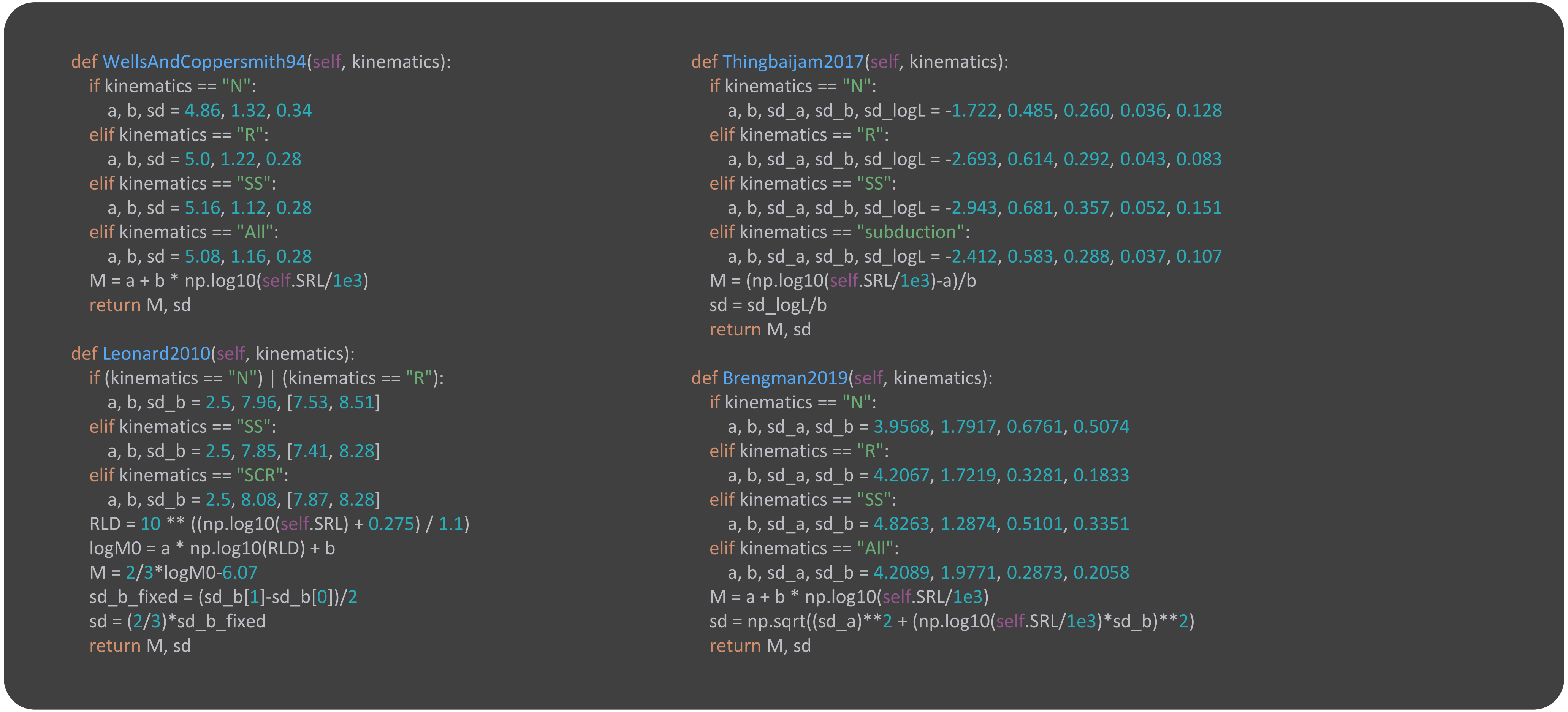
Mw ≥ 6.5 ?



PASO 4

Fault	Length	Length_unc	Mw	Mw_sigma	15.9% quantile
F1	30	3	6.60	0.46	6.19
F2	20	1	6.36	0.46	5.95
F3	8.5	3.5	5.81	0.53	5.30
F4	21	4	6.38	0.47	5.95
F5	140	5	7.47	0.46	7.13
F6	52.5	7.5	6.92	0.47	6.53
F7	25	1	6.49	0.46	6.09
F8	16	2	6.28	0.35	5.93
F9	23	2	6.52	0.35	6.18
F10	15	2	6.23	0.35	5.89
F11	23	1	6.42	0.56	5.98

Relaciones de escala



```
def WellsAndCoppersmith94(self, kinematics):
    if kinematics == "N":
        a, b, sd = 4.86, 1.32, 0.34
    elif kinematics == "R":
        a, b, sd = 5.0, 1.22, 0.28
    elif kinematics == "SS":
        a, b, sd = 5.16, 1.12, 0.28
    elif kinematics == "All":
        a, b, sd = 5.08, 1.16, 0.28
    M = a + b * np.log10(self.SRL/1e3)
    return M, sd

def Leonard2010(self, kinematics):
    if (kinematics == "N") | (kinematics == "R"):
        a, b, sd_b = 2.5, 7.96, [7.53, 8.51]
    elif kinematics == "SS":
        a, b, sd_b = 2.5, 7.85, [7.41, 8.28]
    elif kinematics == "SCR":
        a, b, sd_b = 2.5, 8.08, [7.87, 8.28]
    RLD = 10 ** ((np.log10(self.SRL) + 0.275) / 1.1)
    logM0 = a * np.log10(RLD) + b
    M = 2/3*logM0-6.07
    sd_b_fixed = (sd_b[1]-sd_b[0])/2
    sd = (2/3)*sd_b_fixed
    return M, sd
```

```
def Thingbaijam2017(self, kinematics):
    if kinematics == "N":
        a, b, sd_a, sd_b, sd_logL = -1.722, 0.485, 0.260, 0.036, 0.128
    elif kinematics == "R":
        a, b, sd_a, sd_b, sd_logL = -2.693, 0.614, 0.292, 0.043, 0.083
    elif kinematics == "SS":
        a, b, sd_a, sd_b, sd_logL = -2.943, 0.681, 0.357, 0.052, 0.151
    elif kinematics == "subduction":
        a, b, sd_a, sd_b, sd_logL = -2.412, 0.583, 0.288, 0.037, 0.107
    M = (np.log10(self.SRL/1e3)-a)/b
    sd = sd_logL/b
    return M, sd

def Brengman2019(self, kinematics):
    if kinematics == "N":
        a, b, sd_a, sd_b = 3.9568, 1.7917, 0.6761, 0.5074
    elif kinematics == "R":
        a, b, sd_a, sd_b = 4.2067, 1.7219, 0.3281, 0.1833
    elif kinematics == "SS":
        a, b, sd_a, sd_b = 4.8263, 1.2874, 0.5101, 0.3351
    elif kinematics == "All":
        a, b, sd_a, sd_b = 4.2089, 1.9771, 0.2873, 0.2058
    M = a + b * np.log10(self.SRL/1e3)
    sd = np.sqrt((sd_a)**2 + (np.log10(self.SRL/1e3)*sd_b)**2)
    return M, sd
```