1 Matematická formulace problému

Mám řešit problém hledání epicentra zemětřesení ze znalosti L-vln zjištěných ze seismografu. Neboli znám časy t_1, t_2, t_3 ve kterých naměřili jednotlivé stanice signál a polohy $x_1 = (\theta_1, \phi_1)$, $x_2 = (\theta_2, \phi_2)$, $x_3 = (\theta_3, \phi_3)$ těchto stanic. Vzdálenost dvou bodů na sféře se určí jako

$$d(x_i = (\theta_i, \phi_i), x_j = (\theta_j, \phi_j)) = R \arccos\left[\cos\theta_i \cos\theta_j \cos(\phi_i - \phi_j) + \sin\theta_i \sin\theta_j\right] \tag{1}$$

Označím si polohu epicentra jako $s = (\theta_s, \phi_s)$. Vzhledem k tomu, že předpokládáme stejnou rychlost šíření vlny ve všech směrech, platí mezí vzdálenostmi stanic a epicentra.

$$d(x_1, s) = vt_1 \tag{2}$$

$$d(x_2, s) = vt_2 \tag{3}$$

$$d(x_3, s) = vt_3 \tag{4}$$

Z rovnic vyloučím v podělením rovnic.

$$\frac{d(x_1, s)}{d(x_2, s)} = \frac{t_1}{t_2} \tag{5}$$

$$\frac{d(x_1, s)}{d(x_3, s)} = \frac{t_1}{t_3} \tag{6}$$

Problém, který budu řešit, bude nelineární soustava dvou rovnic o dvou neznámých $s = (\theta_s, \phi_s)$.

$$\frac{d(x_1, s)}{d(x_2, s)} - \frac{t_1}{t_2} = 0 \tag{7}$$

$$\frac{d(x_1, s)}{d(x_3, s)} - \frac{t_1}{t_3} = 0 \tag{8}$$

2 Popis použité numerické metody

K řešení využiji Newton-Raphsonovu metodu. Obecně hledám-li řešení soustavy rovnic

$$\begin{aligned} f_1(x_1, ..., x_n) &= 0 \\ &... \\ f_n(x_1, ..., x_n) &= 0 \end{aligned} \tag{9}$$

nebo úspornější, pokud $\mathbf{x} = (x_1, ..., x_n)^T$ a $\mathbf{f}$ je vektor funkcí f_i

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{10}$$

pak pro zpřesnění odhadu $\mathbf{x}_n$ platí

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{J}_f^{-1}(\mathbf{x}_n)\mathbf{f}(\mathbf{x}_n) \tag{11}$$

Přitom $\mathbf{J}_f$ je jakobián funkcí $\mathbf{f}$ a dá se určit jako

$$\mathbf{J}_f = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \tag{12}$$

3 Zdrojový kód

Nejdříve si naimportuji potřebné symboly a nadefinuji konstantu pro poloměr zeměkoule.

```python
from numpy import arctan2, arccos, arcsin, cos, sin, matrix, subtract, deg2rad, rad2deg, multiply, sqrt
EARTH_RADIUS = 6371000
```

Nyní si vytvořím pomocnou třídu na reprezentaci polohy na sféře v daném čase. Na ní si zároveň definuji instanční metodu *d*, která jako argument přijímá jiný objekt stejného typu a určí vzdálenost mezi těmito místy na sféře.

```python
class EarthPosition(object):
    def __init__(self, latitude, longitude, time=0):
        self.latitude = deg2rad(latitude)
        self.longitude = deg2rad(longitude)
        self.time = time

    def d(a, b):
        return EARTH_RADIUS * arccos(
            cos(a.latitude) * cos(b.latitude) * cos(a.longitude - b.longitude) + sin(a.latitude) * sin(b.latitude)
        )

    def __repr__(self):
        return "[{0}, {1}]".format(rad2deg(self.latitude), rad2deg(self.longitude))
```

Dále definuje pomocnou funkci pro numerickou derivaci funkce.

```python
def derivative(fun, x):
    h = 1e-6
    return (fun(x + h) - fun(x - h)) / (2 * h)
```

Nakonec samotná třída, která hledá polohu epicentru. V konstruktoru je potřeba ji předat trojici objektů typu *EarthPosition*. Funkce *test_function* představuje každou ze dvou funkcí z rovnice $f_i(x_1, x_2, \mathbf{x}) = 0$ (obě rovnice mají stejný tvar, až na vstupní parametry x_i a t_i). Funkce *longitude_function* a *latitude_function* jsou pomocné funkce vyššího řádu, které generují funkce jednoho argumentu - jedné ze zeměpisných souřadnic. Zadefinoval jsem je, abych mohl jednoduše použít funkci pro numerickou derivaci funkce jedné proměnné.

Metoda *inverse_jacoby* počítá inverzní Jacobián, který se použije v metodě *solve*. Tato metoda provádí samotné iterace Newtonovy metody, v každé iteraci se určí jacobián a vektor funkcí *f_matrix*, skalárně se vynásobí a z tohoto vektoru se určí nová oprava pozice.

```python
class EpicenterSolver(object):
    def __init__(self, x1, x2, x3):
        self.x1 = x1
        self.x2 = x2
        self.x3 = x3

    @classmethod
    def test_function(cls, x1, x2, x):
        return x1.d(x) / x2.d(x) - x1.time / x2.time

    @classmethod
    def longitude_function(cls, x1, x2, latitude):
        return lambda x: cls.test_function(x1, x2, EarthPosition(latitude, x))

    @classmethod
    def latitude_function(cls, x1, x2, longitude):
        return lambda x: cls.test_function(x1, x2, EarthPosition(x, longitude))

```

```python
    def inverse_jacoby(self, x):
        f1_lat = self.latitude_function(self.x1, self.x2, x.longitude)
        f1_lon = self.longitude_function(self.x1, self.x2, x.latitude)

        f2_lat = self.latitude_function(self.x2, self.x3, x.longitude)
        f2_lon = self.longitude_function(self.x2, self.x3, x.latitude)

        a = derivative(f1_lat, x.latitude)
        b = derivative(f1_lon, x.longitude)
        c = derivative(f2_lat, x.latitude)
        d = derivative(f2_lon, x.longitude)

        return multiply(1 / (a * d - b * c), matrix([[d, -b], [-c, a]]))

    def solve(self, s, n):
        """Solve the system of equation by Newton-Raphson method."""

        def iteration(x):
            inverse_jacobi = self.inverse_jacoby(x)
            f_matrix = matrix([
                [self.test_function(self.x1, self.x2, x)],
                [self.test_function(self.x2, self.x3, x)],
            ])
            product = inverse_jacobi.dot(f_matrix)
            return EarthPosition(x.latitude - product.item(0), x.longitude - product.item(1))

        for _ in range(n):
            s = iteration(s)

        return s
```

Nakonec počítám epicentrum pro hodnoty odečtené z grafu - toto by šlo samozřejmě napsat obecně a vstup přijímat např. jako CLI argumenty nebo vstup ze souboru, pro moje účelý to ale bylo zbytečné. Počáteční odhad jsem zvolil naivně jako průměr zadaných zeměpisných souřadnic.

```python
def starting_estimate(x1, x2, x3):
    return EarthPosition(
        (x1.latitude + x2.latitude + x3.latitude) / 3,
        (x1.longitude + x2.longitude + x3.longitude) / 3,
    )


if __name__ == "__main__":
    x1 = EarthPosition(61.601944, -149.117222, 7.5) # Palmer, Alaska
    x2 = EarthPosition(39.746944, -105.210833, 23) # Golden, Colorado
    x3 = EarthPosition(4.711111, -74.072222, 44) # Bogota, Columbia

    solver = EpicenterSolver(x1, x2, x3)
    solution = solver.solve(starting_estimate(x1, x2, x3), 1000)
    print(solution)
```

4 Výsledky

Po spuštění dostanu výsledek.

```
bash\$ python app.py
[51.8347986915, 54.5086392079]
```

Po porovnání vzdáleností s grafem dostávám značné odchylky ve vzdálenostech epicentra a stanic. Provedl jsem ještě jeden test přidáním testovací funkce na zjištění konvergence při malých odchylkách souřadnic a podle výsledků se metoda v pořádku odchyluje pouze a malé hodnoty souřadnic. Provedl jsem i testování s jinou funkcí vzdálenosti na sféře, zde ale problém také není.

```python
def stability_testing(last_solution):
    for x in range(0, 10):
        for y in range(0, 10):
            x1 = EarthPosition(61.601944 + x * 0.1, -149.117222 + y * 0.1, 7.5) # Palmer, Alaska
            x2 = EarthPosition(39.746944, -105.210833, 23) # Golden, Colorado
            x3 = EarthPosition(4.711111, -74.072222, 44) # Bogota, Columbia
            solver = EpicenterSolver(x1, x2, x3)
            solution = solver.solve(starting_estimate(x1, x2, x3), 1000)
            print(EarthPosition(
                solution.latitude - last_solution.latitude,
                solution.longitude - last_solution.longitude,
                0
            ))
```

Nakonec jsem při testování zjistil, že i malé odchylky pro zeměpisné souřadnice zadaných stanice vyvolají rozdíli i tisíce mil pro výsledné vzdálenosti stanic a epicentra - získané nepřesnosti jsou tedy pravděpodobny způsobeny nepřesnostmi v zadaných polohách a časech (kvůli velkému poloměru země).