NISMOD

Hands-On 7 – Testing multiple points of failure

This exercise explores the risks of multiple failures across the road network, introducing a shortest-path routing model to demonstrate how single or multiple road link failures can have indirect impacts on the way people can use the network.

Learning outcomes

- Assess direct and some indirect impacts of multiple asset failures
- Compare flooding within regions as source of multiple failure scenarios
- Understand approaches to stress-testing the system under multiple failures.

Map exposure within regions

Open the notebook "03-test-multiple-failures.ipynb". This contains code to implement each of the steps in this tutorial.

Read in the road network (both edges and nodes) and regions, then use the geopandas "sjoin" method to add region name and code to the roads data.

```
roads = gpd.sjoin(roads, regions) \

.drop(columns='index_right')
```

Filter roads by region name to find all roads in Greater Accra:

```
accra_roads = roads[roads.ADM1_EN == 'Greater Accra']
```

Read in all road exposure, then filter by region, RCP and return period to find all roads exposed to a historical 100-year flood in Greater Accra.

Multiple failures

Direct damage can be summed directly, if we assume that all roads are damaged in the same event:

```
accra_exposure.damage_usd.sum()
```

Indirect damage can be assessed in different ways, some beyond the scope of this notebook. In this section, we look at the effects of disruption on a single route across the Greater Accra region. In a fuller analysis, we could extend this to look at many trips made within the region and calculate the number of passengers or value of freight disrupted, along with the effects on transport time and cost.

Start by creating a networkx graph from the roads, using from_id, to_id and length_m:

```
G = nx.Graph()

G.add_edges_from((r.from_id, r.to_id, {"id": r.id, "weight":
r.length_m}) for r in roads.itertuples())
```

Then find the shortest path from one node to another:

```
route_nodes = nx.algorithms.shortest_path(G, "roadn_6700",
"roadn_1011", weight='weight')
```

Then, using this list of nodes, find the corresponding road edges that make up the shortest path, and sum over their lengths to find the length of the whole route.

```
def edge_ids_from_nodes(G, route_nodes):
    next_nodes = iter(route_nodes)
    next(next_nodes)
    return [
        G.edges[u, v]['id']
        for u, v in zip(route_nodes, next_nodes)
    ]
route_edge_ids = edge_ids_from_nodes(G, route_nodes)
route = roads[roads.id.isin(route_edge_ids)]
```

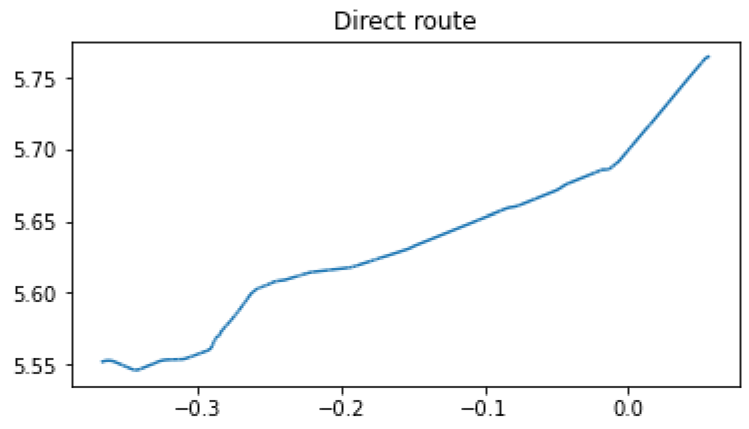

Figure 7.1: Direct route from node 6700 to node 1011 (axes are latitude and longitude)

The notebook already contains the definition of a “calc_route” function to repeat this analysis under different conditions of failure. Run the following cells to test each of the scenarios described in the next few paragraphs.

Test a single road failure to find if disruption makes a difference to the overall route:

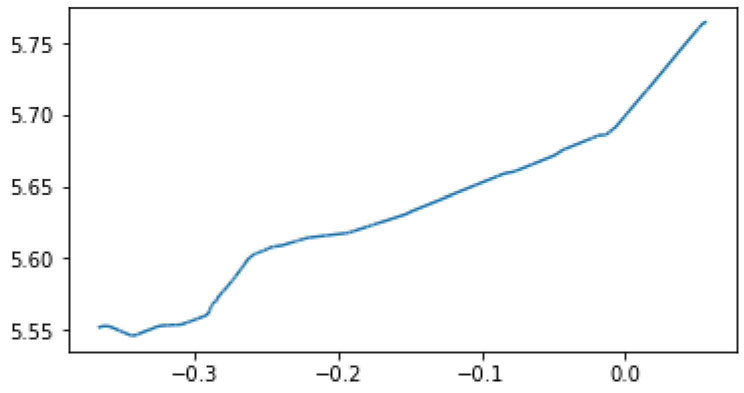

Figure 7.2: Shortest route from node 6700 to node 1011 under single edge failure.

The single road failure above has almost no effect. In our dataset, the lanes of this road are represented separately, so the routing algorithm finds a route which goes around the failed link by switching to the other lane, and the whole journey is only about 10m longer.

Test the effect of both lanes flooded at the same time, which may be more realistic:

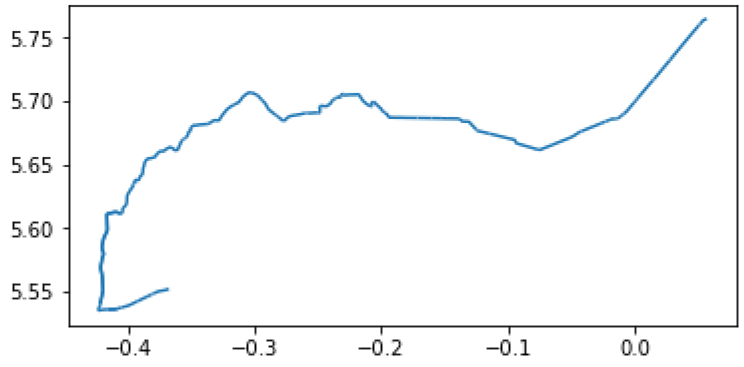

Figure 7.3: Shortest route from node 6700 to node 1011 when both lanes fail.

This results in a much longer route around the flooded link.

What if more than one road is disrupted at the same time? Test what happens if we assume that all roads exposed to any 100-year flood event anywhere in Greater Accra are impassible.

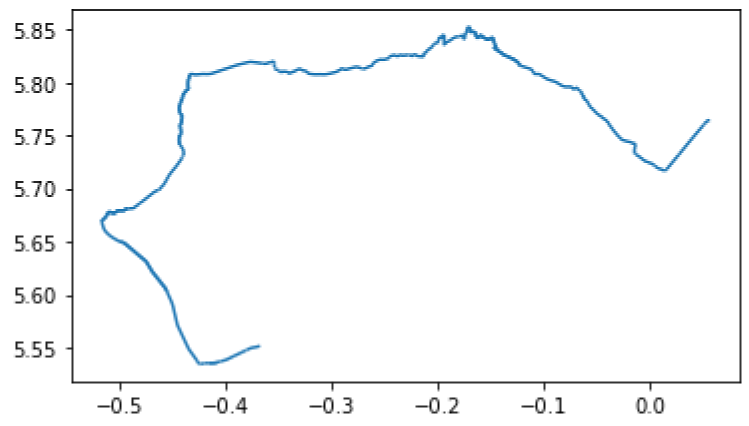

Figure 7.4: Shortest route from node 6700 to node 1011 when all roads exposed to any 100-year flood event are impassible.

This gives a longer route again.

This is a quick way of coming up with a hypothetical flood event, but it is not a rigorous method of analysis. With historic flood outlines, we could test and validate this simple model against past events. With an event-set output from a hydrological model (rather than just the return-period hazard map that we've been using), we could test an ensemble of potential events.

The next section looks at testing all possible combinations of failures, which doesn't require any additional data or modelling.

Test combinations

We can calculate the number of possible combinations of failures, and it gets very large quite quickly.

For example, with three roads, {A, B, C}, there are three possible single failures ({only A}, {only B} or {only C}), three possible double failures ({A and B}, {B and C} or {A and C}), and one possible triple failure ({A, B and C}).

More formally, if a set has *n* elements, the number of ways of picking *k* elements can be shown to be:

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots 1} = \frac{n!}{k!(n-k)!}$$

Equation (Figure) 7.5: Number of combinations is equivalent to the binomial coefficient

and is zero when $k > n$.

Define this function in code using “factorial” from the math module in the Python standard library (or use the definition provided in the notebook) and try a few values to see how it behaves. Calculate some of the numbers of possible failure combinations within our road network.

```
n = len(roads)
print(f"With {n} roads")
for k in range(4):
    print(f"there are {n_choose_k(n, k):,} total possible combinations
of {k} roads failing")
```

We can also use the np.random.choice method to sample failure combinations at random from all roads (regardless of whether they intersect with any hazard).

Sample 100 different sets of 500 failures to test how the best route length (for this arbitrarily chosen route) changes under random failure conditions, then use the pandas “describe” method to calculate basic summary statistics from this sample.

```
k = 500
n_samples = 100
lengths = []
for _ in tqdm(range(n_samples)):
    ids = np.random.choice(roads.id, size=k, replace=False)
    failed_roads = roads[roads.id.isin(ids)]
    random_failures = [
        (road.from_id, road.to_id)
        for road in failed_roads.itertuples()
    ]
    random_fail_route = calc_route(roads, random_failures,
"roadn_6700", "roadn_1011")
    length = round(random_fail_route.length_m.sum() / 1e3, 1)
    lengths.append(length)
```

Plot all the route lengths as a scatter plot, to get some visual idea of the distribution. Figure 7.6 shows the result of one set of random runs, where most results are clustered around 55-60km, close to the shortest route when there are no failures.

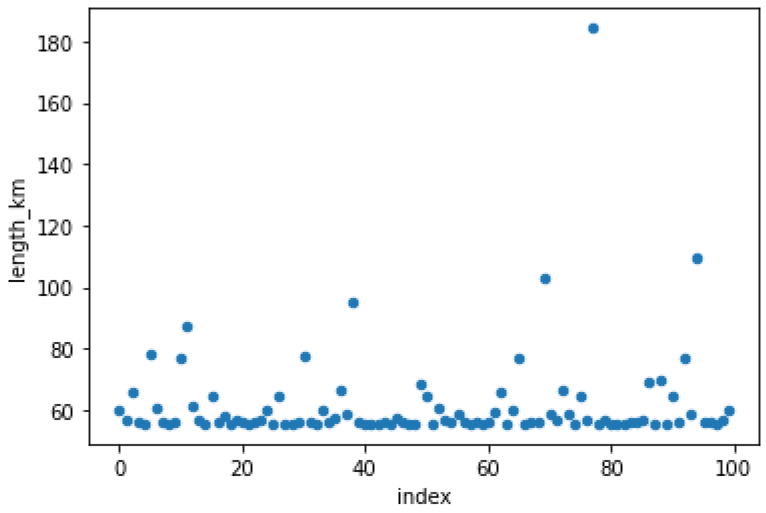

Figure 7.6: Scatter plot of route lengths under random failure scenarios.

Plot the empirical cumulative distribution function to summarise the distribution in another way. Figure 7.7 below shows the same data as figure 7.6 above, here with route length on the x axis and cumulative probability on the y axis. From the samples we've taken while testing this tutorial, it shows that most of the time, 500 random failures in the road network have little effect on the route length (around 55-60km), but some combinations of failures see a route length of up to around 180km.

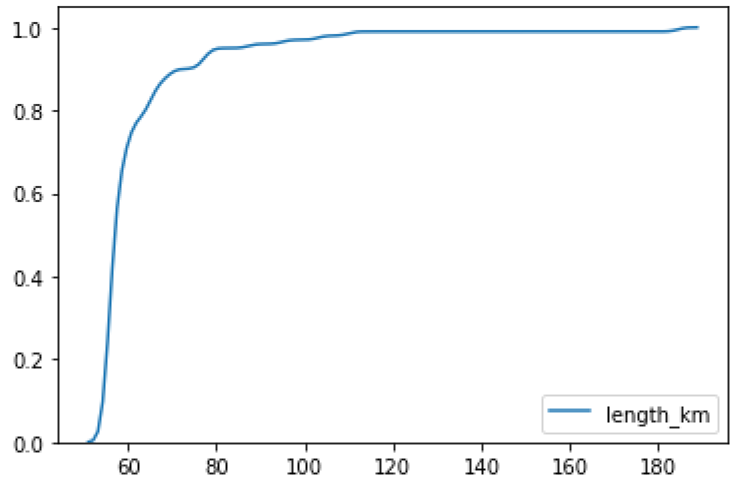

Figure 7.7: Empirical cumulative distribution function of route lengths under random failure scenarios.

As an extension exercise, try using the geopandas "to_file" method to save any of the route GeoDataFrames to a file, and open in QGIS to explore alongside the other mapped datasets. Adapt the code below to save the route you're interested in:

```
route.to_file(
    os.path.join(data_folder, 'route.gpkg',
    driver='GPKG'
)
```

Summary

In this session we selected all exposure within the Greater Accra region to sum up the direct damages if any road exposed to any historical 100-year flood event were damaged at the same time. We then introduced an example route across the region, finding the shortest path from A to B under normal conditions, then testing how it would change under different failure scenarios.