Modelling Transportation Networks with Octave

Six Silberman

June 12, 2008

Abstract

This document presents and discusses some code for modelling transportation networks using the Octave language/environment. Specifically included is code for performing all-or-nothing assignment, capacity-restrained assignment, incremental assignment, iterative assignment with the method of successive averages, and user equilibrium assignment with the Franke-Wolfe algorithm. All ancillary code that is not included with the Octave core (e.g., Dijkstra's algorithm, path accounting) is also included and discussed. Runs with a simple toy network are detailed for each assignment technique, and a user equilibrium run via Frank-Wolfe on the 'Tromaville' network is presented.

NOTE All included code is compatible with MATLAB, with the possible exception of the call to `quadl` in `ueof.m`. In Octave, the last two parameters in this call are ignored by `quadl` and passed directly to the `bpr` function, which is used to compute travel times.

Contents

1 The Problem

The problem is the general user equilibrium problem; that is, the optimization problem

$$\min \sum \int_0^x t_a(\omega) d\omega \tag{1}$$

subject to

$$\begin{aligned} \sum_k f_k^{od} &= T_{od} && (2) \\ x_a &= \sum_o \sum_d \sum_k f_k^{od} \delta_{ak}^{od} && (3) \\ f_k^{od} &\geq 0 && (4) \end{aligned}$$

for origin nodes o, destination nodes d, flows f on paths k, total origin-destination demand T^{od}, and

$$\delta_{ak}^{od} = \begin{cases} 1, \text{ if link } a \text{ is on path } k \text{ between origin } o \text{ and destination } d \\ 0, \text{ otherwise} \end{cases} \tag{5}$$

2 Important Note

In Octave and MATLAB, array indices begin at 1, not 0.

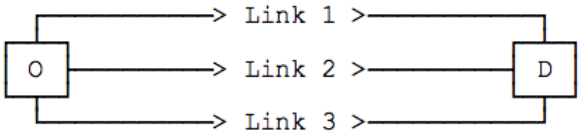
3 All-or-Nothing Assignment and Modelling Infrastructure

In all-or-nothing assignment, all demand from node i to node j is assigned to the shortest path from i to j. In the context of solving the user equilibrium problem, this is equivalent to assuming that there are no congestion effects.

The following steps are performed:

1. Input the network representation, including information on link travel times and capacities.
2. Compute the matrix of shortest travel times from all nodes to all nodes in the network (the 'skim matrix').
3. Construct the 4-dimensional object δ_{ak}^{od} that keeps track of which paths between which nodes comprise which links.
4. Assign the total O-D demand to each *link* on the shortest path between all origin and destination nodes.

Consider this simple network from McNally 2006, 'Sample Application of Traffic Assignment Techniques.'



with a total O-D demand T^{od} of 10 vehicles per hour (vph). Base travel times on links 1, 2, and 3 are 10, 20, and 25 minutes respectively, and capacities are 2, 4, and 5 respectively. The link performance function is the BPR LPF

$$t(v) = t_0 \left[1 + \alpha \left(\frac{v}{c}\right)^{\beta}\right] \tag{6}$$

where t is the travel time on the link (in minutes), $\alpha = 0.15$, $\beta = 4$, v is the volume on the link, and c is the 'capacity' of the link. As indicated in the diagram, all links are unidirectional.

Although conceptually simple, the graph which represents this network is a multigraph (i.e., a graph with multiple edges between node pairs), which can present computational

difficulties for software prepared with simple graphs in mind. Fortunately, the multigraph can be represented fairly simply as a network with 5 nodes: the origin node becomes 'node 1', the destination 'node 5', and 3 intermediate nodes are introduced, splitting each link into two link with half the travel time of the original (but the same capacity). The network can be represented in 'augmented ladder' form as

```
1  2  5     2
2  5  5     2
1  3  10    4
3  5  10    4
1  4  12.5  3
4  5  12.5  3
```

where each row contains information about a separate link: the first column is the origin node, the second column the destination node, the third column the base travel time t_0, and the fourth column the 'capacity' c. This can be entered into Octave as follows:

```
ladder = [1 2 5 2; 2 5 5 2; 1 3 10 4; 3 5 10 4; 1 4 12.5 3; 4 5 12.5 3];
```

The O-D demand matrix is very simple: there are 10 units of demand from node 1 to node 5, and no demand between any other node pairs. In Octave this can be entered simply:

```
od_demand = zeros(5, 5);
od_demand(1, 5) = 10;
```

This enters the matrix

```
0    0    0    0   10
0    0    0    0    0
0    0    0    0    0
0    0    0    0    0
0    0    0    0    0
```

into the variable `od_demand`.

Although conceptually simple, the ladder representation is less handy than a matrix representation for computational purposes. A function `build_skeleton` can perform the conversion:

```
1     function [skeleton, capacities] = build_skeleton(ladder, bidirectional)

      n = max(max(ladder(:, 1:2)));

      skeleton = zeros(n, n);
      capacities = zeros(n, n);

5     for i = 1:length(ladder)
        skeleton(ladder(i, 2), ladder(i, 1)) = ladder(i, 3);
        capacities(ladder(i, 2), ladder(i, 1)) = ladder(i, 4);
        if (bidirectional == 1)
          skeleton(ladder(i, 1), ladder(i, 2)) = ladder(i, 3);
10        capacities(ladder(i, 1), ladder(i, 2)) = ladder(i, 4);
        end
      end

      skeleton(find(skeleton == 0)) = inf;
      for i = 1:n
15      skeleton(i, i) = 0;
      end
```

The first line is the Octave function definition: this is the function `build_skeleton`, which outputs variables `skeleton` and `capacities` after receiving as input the variables `ladder` and `bidirectional`. `ladder` is the augmented ladder representation of the network discussed above; `bidirectional` is simply a flag, set to 0 ('false') or 1 ('true') indicating whether all links should be assumed bidirectional or not.

Line 2 infers the number of nodes n in the network by finding the largest value (node number) in the first two columns of the augmented ladder representation. Lines 3 and 4 initialize the output variables; they are both n-by-n matrices.

The `for` loop on lines 5-12 traverses the ladder representation, adding links to the matrix representation. If the call to `build_skeleton` indicated that the ladder should be taken to represent a network where all links are bidirectional, the `if` clause on lines 8-11 will add the appropriate links to the matrix, resulting in symmetric matrices `skeleton` and `capacities`.

Finally, lines 13-16 set all entries in `skeleton` corresponding to nonexistent links to `inf` ('infinity' in Octave), and all entries on the diagonal to zero. Put another way, if nodes i and j are not directly connected, the 'skeleton' distance between them is infinity (even though it may be possible to travel between them through intermediate nodes); similarly, the distance between a node and itself is zero.

The call `[skeleton, capacities] = build_skeleton(ladder, 0)`, with `ladder` as above, yields the following matrix in `skeleton`:

```
    0.00000        Inf        Inf        Inf        Inf
    5.00000    0.00000        Inf        Inf        Inf
   10.00000        Inf    0.00000        Inf        Inf
   12.50000        Inf        Inf    0.00000        Inf
        Inf    5.00000   10.00000   12.50000    0.00000
```

and in `capacities`:

```
   0   0   0   0   0
   2   0   0   0   0
   4   0   0   0   0
   3   0   0   0   0
   0   2   4   3   0
```

After the network is input in this way, the skim matrix can be computed:

```
function [skims, back_nodes] = generate_skims(skeleton)

n = length(skeleton);
back_nodes = zeros(n, n);
skims = zeros(n, n);
visited = zeros(n, n);

for i = 1:n
  [skims(:, i), back_nodes(:, i), visited(:, i)] = dijkstra(skeleton, i);
end

skims = skims';
```

Two n-by-n matrices are output, `skims` and `back_nodes`. `skims` is the skim matrix; its i, jth entry contains the minimum travel time from node i to node j. Entries i, j of `inf` indicate that it is not possible to travel from node i to node j. `back_nodes` contains the shortest paths given these skims; its i, jth entry is the back node of node j on the shortest path from node i to node j. Entries of `nan` (other than those on the diagonal) indicate that it is not possible to travel to node j from node i; therefore no back node exists.

Line 9 simply transposes the `skims` matrix.

The call `[skims, back_nodes] = generate_skims(skeleton)`, with `skeleton` from above, yields the following matrix in `skims`:

```
    0.00000         Inf         Inf         Inf         Inf
    5.00000     0.00000         Inf         Inf         Inf
   10.00000         Inf     0.00000         Inf         Inf
   12.50000         Inf         Inf     0.00000         Inf
   10.00000     5.00000    10.00000    12.50000     0.00000
```

and in `back_nodes`:

```
   NaN     2     3     4     2
   NaN   NaN   NaN   NaN     5
   NaN   NaN   NaN   NaN     5
   NaN   NaN   NaN   NaN     5
   NaN   NaN   NaN   NaN   NaN
```

At the heart of the function `generate_skims` (specifically on line 7) is another function, `dijkstra`, which uses Dijkstra's algorithm to compute the skim trees for each node i:

```
1    function [distances, previous, visited] = dijkstra(graph, s)

     n = length(graph);

     distances = inf(n, 1);
     previous = nan(n, 1);
5    visited = zeros(n, 1);

     c = s;
     distances(s) = 0;

     while !all(visited)
       visited(c) = 1;
10     for j = 1:n
         if (graph(c, j) != 0) && (graph(c, j) != inf)
           if distances(c) + graph(c, j) < distances(j)
             distances(j) = distances(c) + graph(c, j);
             previous(j) = c;
15         end
         end
       end
       nv = nan;
       for j = 1:n
20       if (visited(j) == 0)
           if isnan(nv)
              nv = j;
           end
           if distances(j) < distances(nv)
25           nv = j;
           end
         end
       end
       c = nv;
30   end
```

This is a straightforward implementation of Dijkstra's algorithm. The vectors `distances` and `previous` form the matrices `skims` and `back_nodes` output by `generate_skims`, discussed above. The vector `visited` is used by the algorithm to keep track of which nodes have been visited and which have not; after the algorithm terminates, all its entries are 1.

The tensor δ_{ak}^{od} which stores the links on the shortest paths can be computed from the `back_nodes` matrix. Here, because links are identified only by the nodes they connect (this includes their directionality), a 4-dimensional object δ_{ij}^{rs} is constructed which identifies only the links on the shortest path between each node pair. Specifically, its i, j, r, sth entry is 1 if the link from node r to node s is on the shortest path from node i to node j, and 0 otherwise:

```
function delta = build_delta(back_nodes)

n = length(back_nodes);
delta = zeros(n, n, n, n);

for i = 1:n
  for j = 1:n
    last = i;
    while !isnan(back_nodes(last, j))
      delta(i, j, last, back_nodes(last, j)) = 1;
      last = back_nodes(last, j);
    end
  end
end
```

This function traverses the `back_nodes` matrix, recursing to the start of the path from node i to node j at *each* entry i, j, setting the appropriate entry of δ_{ij}^{rs} to 1 with each step until the origin node is reached. The Octave function `isnan` returns true if its argument is `nan` ("not a number"), the Octave equivalent of `null`; the `while` loop beginning on line 7 breaks when it recurses to the origin node.

The call `delta = build_delta(back_nodes)`, with `back_nodes` from above, yields a 4-dimensional object with 625 entries. These are all zero, except i, j, r, s-entries

```
1, 2, 1, 2
1, 3, 1, 3
1, 4, 1, 4
1, 5, 1, 2
1, 5, 2, 5
2, 5, 2, 5
3, 5, 3, 5
4, 5, 4, 5
```

which are all 1. For example, '`1, 5, 1, 2`' (line 4) indicates that the link from node 1 to node 2 is on the shortest path from node 1 to node 5.

After δ_{ij}^{rs} (here `delta`, on line 3) is constructed, the assignment can proceed:

```
function link_flows = aon(skeleton, od_demand)

[skims, back_nodes] = generate_skims(skeleton);
delta = build_delta(back_nodes);

n = length(skeleton);
link_flows = zeros(n, n);
p = length(od_demand);

for i = 1:p
  for j = 1:p
    for k = 1:n
      for m = 1:n
        if (delta(i, j, k, m) == 1)
          link_flows(k, m) = link_flows(k, m) + od_demand(i, j);
        end
      end
    end
  end
end

link_flows = link_flows';
link_flows(find(skeleton == inf)) = nan;
```

This function simply traverses the subset of entries in δ_{ij}^{rs} which correspond to legitimate O-D pairs r, s (here the notation is somewhat confusing; the variables used to traverse these node pairs in the code are `i` and `j`, lines 7-8), adding the total O-D demand to every link on the shortest path (line 11-12). Line 18 transposes the `link_flows` matrix, and line 19 sets all entries corresponding to non-adjacent node pairs to `nan`.

The call `link_flows = aon(skeleton, od_demand)` with `skeleton` and `od_demand` as above, yields the set of flows (in the matrix `link_flows`)

```
     0   NaN   NaN   NaN   NaN
    10     0   NaN   NaN   NaN
     0   NaN     0   NaN   NaN
     0   NaN   NaN     0   NaN
   NaN    10     0     0     0
```

That is, all traffic is assigned to the links from node 1 to node 2 and from node 2 to node 5. Entering this value into the BPR LPF, we observe that each link has a travel time of 473.75 minutes, for a total path travel time of 947.5 minutes.

4 Capacity-Restrained Assignment

All-or-nothing assignment is obviously unrealistic as a method for modelling actual traffic, as its modelled link flows are unaffected by congestion and therefore cannot make use of link performance models. A simple attempt to model congestion effects might proceed by making use of successive AON assignments. This is the naïve capacity-restrained assignment (CRA) method (referred to in McNally 2006, *op. cit.* as 'CRA1', or 'Basic Capacity Restrained Assignment'). After constructing the network representation, computing the skim trees, and constructing δ_{ak}^{od} (or, as here, δ_{ij}^{rs}), the algorithm performs repeated AON assignments, updating the link travel times and skim matrix at each iteration. The algorithm stops when it converges or when a predetermined maximum number of iterations is reached; CRA1 should not be expected to converge for the vast majority of networks, and indeed probably for all real networks.

The function is as follows:

```
function [link_flows, link_times] = cra(ladder, bidirectional, od_demand, iters)

[skeleton, capacities] = build_skeleton(ladder, bidirectional);

n = length(skeleton);
link_flows = zeros(n, n, iters);

link_times = zeros(n, n, iters + 1);
link_times(:, :, 1) = skeleton;

for i = 1:iters
  link_flows(:, :, i) = aon(link_times(:, :, i), od_demand)
  link_times(:, :, i + 1) = ...
    update_link_times(skeleton, capacities, squeeze(link_flows(:, :, i)))
end
```

(Note that the last assignment before the `end` should be one line in the code; it is broken here across two lines.)

The function call takes effectively the same data as the AON assignment, although in the version of AON included above, the network is passed as a `skeleton` rather than in the `ladder` representation with a `bidirectionality` flag, as in the CRA function here.[1] New is the `iters` parameter, which is the (integer) number of iterations to perform before stopping. There is no convergence test coded in, nor would one help much, as CRA is not expected to converge.

[1] `aon` takes the `skeleton` rather than building it from the `ladder` so that `aon` can be called repeatedly by other functions, like `cra`, without wasting too many processing cycles.

Note that `link_flows` and `link_times` are *three*-dimensional objects; all link flows and link travel times at each iteration are stored for later use rather than discarded. (The implementation can be adjusted to discard this data after it is used if memory is at a premium.) Specifically, The i, j, kth entry of `link_flows` is the flow on the link from node i to node j at iteration k, and the i, j, kth entry of `link_times` is the travel time on the link from node i to node j at iteration $k-1$, where 'iteration 0' (i.e., the entries with $k = 1$) is the set of skims obtained from the base travel times (obtained directly from the `skeleton` on line 6).

In the `for` loop on lines 7-10, the algorithm first performs an AON assignment (by calling `aon`, line 8), then updates the link travel times accordingly (by calling `update_link_times`, line 9). The function `update_link_times` computes the new link travel times based on the new link flows:

```
function new_link_times = update_link_times(skeleton, capacities, link_flows)

n = length(skeleton);
new_link_times = zeros(n, n);

alpha = 0.15;
beta = 4;

new_link_times = skeleton .* (1 + (alpha *  (link_flows ./ capacities) .^ beta));

new_link_times(find(isnan(new_link_times))) = inf;
for i = 1:n
  new_link_times(i, i) = 0;
end
```

The function uses the BPR LPF (Eqn. 6); the constants are defined on lines 4-5, and the matrix element-by-element computation is performed all on one line (line 6) using the Octave/MATLAB element-by-element multiplication, division, and exponentiation operators `.*`, `./`, and `.^` respectively.

(Note that $\boldsymbol{C}$ is the 'element-by-element product' of $\boldsymbol{A}$ and $\boldsymbol{B}$ if $C_{ij} = A_{ij}B_{ij}$, where no summation notation is implied; the same is true for matrix element-by-element division and exponentiation.)

The last block of code simply sets entries corresponding to nonexistent links to `inf` (line 7) and entries on the diagonal to zero (lines 8-10).

The call `[link_flows, link_times] = cra(ladder, 0, od_demand, 3)`, with `ladder` and `od_demand` as above, yields the following link flows at each iteration:

```
link_flows(:,:,1) =

     0   NaN   NaN   NaN   NaN
    10     0   NaN   NaN   NaN
     0   NaN     0   NaN   NaN
     0   NaN   NaN     0   NaN
   NaN    10     0     0     0

link_flows(:,:,2) =

     0   NaN   NaN   NaN   NaN
     0     0   NaN   NaN   NaN
    10   NaN     0   NaN   NaN
     0   NaN   NaN     0   NaN
   NaN     0    10     0     0

link_flows(:,:,3) =

     0   NaN   NaN   NaN   NaN
    10     0   NaN   NaN   NaN
     0   NaN     0   NaN   NaN
     0   NaN   NaN     0   NaN
   NaN    10     0     0     0
```

and the following link travel times (recall here that the iteration number is actually $k-1$, for k the third index, and the entries for which $k = 1$ correspond to the 'zeroth iteration'—that is, the base or 'skeleton' travel times):

```
link_times(:,:,1) =

    0.00000        Inf        Inf        Inf        Inf
    5.00000    0.00000        Inf        Inf        Inf
   10.00000        Inf    0.00000        Inf        Inf
   12.50000        Inf        Inf    0.00000        Inf
        Inf    5.00000   10.00000   12.50000    0.00000

link_times(:,:,2) =

     0.00000         Inf         Inf         Inf         Inf
   473.75000     0.00000         Inf         Inf         Inf
    10.00000         Inf     0.00000         Inf         Inf
    12.50000         Inf         Inf     0.00000         Inf
         Inf   473.75000    10.00000    12.50000     0.00000

link_times(:,:,3) =

    0.00000        Inf        Inf        Inf        Inf
    5.00000    0.00000        Inf        Inf        Inf
   68.59375        Inf    0.00000        Inf        Inf
   12.50000        Inf        Inf    0.00000        Inf
        Inf    5.00000   68.59375   12.50000    0.00000

link_times(:,:,4) =

     0.00000         Inf         Inf         Inf         Inf
   473.75000     0.00000         Inf         Inf         Inf
    10.00000         Inf     0.00000         Inf         Inf
    12.50000         Inf         Inf     0.00000         Inf
         Inf   473.75000    10.00000    12.50000     0.00000
```

These flow and link travel time matrices exhibit the 'flip-flopping' (i.e., nonconvergence) characteristic of CRA1.

5 Incremental Assignment

Incremental assignment approaches the problem posed by the nonconvergence of basic CRA by breaking the demand loading process into fractional sections, and updating the skim matrix in between increment loads.

A simple function to perform incremental assignment is as follows:

```
function [link_flows, link_times] = incr(ladder, bidirectional, od_demand, increments)

[skeleton, capacities] = build_skeleton(ladder, bidirectional);

iters = length(increments);

n = length(skeleton);
link_flows = zeros(n, n, iters + 1);

aux_flows = zeros(n, n, iters);

link_times = zeros(n, n, iters + 1);
link_times(:, :, 1) = skeleton;

for i = 1:iters
  aux_flows(:, :, i) = aon(link_times(:, :, i), od_demand * increments(i));
  link_flows(:, :, i + 1) = link_flows(:, :, i) + aux_flows(:, :, i);
  link_times(:, :, i + 1) = ...
    update_link_times(skeleton, capacities, squeeze(link_flows(:, :, i + 1)));
end
```

At each iteration `i`, this function computes a set of auxiliary flows `aux_flows` (line 10) and loads the fraction of total demand corresponding to element `i` of the vector of `increments` (line 11), the sum of which must equal 1. Link travel times are updated, as in CRA (line 12).

The call `[link_flows, link_times] = incr(ladder, 0, od_demand, [.4 .3 .2 .1])`, with `ladder` and `od_demand` as above, yields the following link flows at each iteration. (Note that here the entries with third index k correspond to the values at iteration $k-1$.)

```
link_flows(:,:,1) =

   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0

link_flows(:,:,2) =

     0   NaN   NaN   NaN   NaN
     4     0   NaN   NaN   NaN
     0   NaN     0   NaN   NaN
     0   NaN   NaN     0   NaN
   NaN     4     0     0     0

link_flows(:,:,3) =

     0   NaN   NaN   NaN   NaN
     4     0   NaN   NaN   NaN
     3   NaN     0   NaN   NaN
     0   NaN   NaN     0   NaN
   NaN     4     3     0     0

link_flows(:,:,4) =

     0   NaN   NaN   NaN   NaN
     4     0   NaN   NaN   NaN
     5   NaN     0   NaN   NaN
     0   NaN   NaN     0   NaN
   NaN     4     5     0     0

link_flows(:,:,5) =

     0   NaN   NaN   NaN   NaN
     4     0   NaN   NaN   NaN
     5   NaN     0   NaN   NaN
     1   NaN   NaN     0   NaN
   NaN     4     5     1     0
```

and the link travel times are as follows:

```
link_times(:,:,1) =

    0.00000        Inf        Inf        Inf        Inf
    5.00000    0.00000        Inf        Inf        Inf
   10.00000        Inf    0.00000        Inf        Inf
   12.50000        Inf        Inf    0.00000        Inf
        Inf    5.00000   10.00000   12.50000    0.00000

link_times(:,:,2) =

    0.00000        Inf        Inf        Inf        Inf
   17.00000    0.00000        Inf        Inf        Inf
   10.00000        Inf    0.00000        Inf        Inf
   12.50000        Inf        Inf    0.00000        Inf
        Inf   17.00000   10.00000   12.50000    0.00000

link_times(:,:,3) =

    0.00000        Inf        Inf        Inf        Inf
   17.00000    0.00000        Inf        Inf        Inf
   10.47461        Inf    0.00000        Inf        Inf
   12.50000        Inf        Inf    0.00000        Inf
        Inf   17.00000   10.47461   12.50000    0.00000

link_times(:,:,4) =

    0.00000        Inf        Inf        Inf        Inf
   17.00000    0.00000        Inf        Inf        Inf
   13.66211        Inf    0.00000        Inf        Inf
   12.50000        Inf        Inf    0.00000        Inf
        Inf   17.00000   13.66211   12.50000    0.00000

link_times(:,:,5) =

    0.00000        Inf        Inf        Inf        Inf
   17.00000    0.00000        Inf        Inf        Inf
   13.66211        Inf    0.00000        Inf        Inf
   12.52315        Inf        Inf    0.00000        Inf
        Inf   17.00000   13.66211   12.52315    0.00000
```

6 Iterative Assignment via Method of Successive Averages

Incremental assignment can perform well with careful selection (often by trial and error) of the increments. Appropriate demand partitioning is heavily dependent on the network, and a more robust approach is hardly uncalled for. One such is iterative assignment, which generates a set of auxiliary flows with the entire demand load via AON at each iteration, then performs a convex combination with the link flows at the previous iteration in order to obtain a new set of flows. The central step is

$$V_a^n = (1 - \phi)V_a^{n-1} + \phi F_a^n \tag{7}$$

where V_a^n is the set of link flows for all links a at iteration n, F_a^n is the set of auxiliary flows generated by the AON assignment given the link travel times at iteration n, and ϕ is a weight parameter between 0 and 1.

One iterative algorithm, the Method of Successive Averages (MSA), sets $\phi = \frac{1}{n}$. A simple implementation follows:

```
function [link_flows, link_times] = msa(ladder, bidirectional, od_demand, iters)

[skeleton, capacities] = build_skeleton(ladder, bidirectional);

n = length(skeleton);
link_flows = zeros(n, n, iters + 1);

aux_flows = zeros(n, n, iters);

link_times = zeros(n, n, iters + 1);
link_times(:, :, 1) = skeleton;

for i = 1:iters
  phi = 1 / i;
  aux_flows(:, :, i) = aon(link_times(:, :, i), od_demand);
  link_flows(:, :, i + 1) = ...
    link_flows(:, :, i) * (1 - phi) + aux_flows(:, :, i) * phi;
  link_times(:, :, i + 1) = ...
    update_link_times(skeleton, capacities, squeeze(link_flows(:, :, i + 1)));
end
```

The value of ϕ is set on line 9, and the computation of auxiliary flows (line 10) and the convex combination (line 11) immediately follow.

The call `[link_flows, link_times] = msa(ladder, 0, od_demand, 6)`, with `ladder` and `od_demand` as above, yields the following link flows at each iteration. (Recall that the entries with third index k correspond to the values at iteration $k-1$.)

```
link_flows(:,:,1) =

   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0

link_flows(:,:,2) =

     0   NaN   NaN   NaN   NaN
    10     0   NaN   NaN   NaN
     0   NaN     0   NaN   NaN
     0   NaN   NaN     0   NaN
   NaN    10     0     0     0

link_flows(:,:,3) =

     0   NaN   NaN   NaN   NaN
     5     0   NaN   NaN   NaN
     5   NaN     0   NaN   NaN
     0   NaN   NaN     0   NaN
   NaN     5     5     0     0

link_flows(:,:,4) =

   0.00000       NaN       NaN       NaN       NaN
   3.33333   0.00000       NaN       NaN       NaN
   3.33333       NaN   0.00000       NaN       NaN
   3.33333       NaN       NaN   0.00000       NaN
       NaN   3.33333   3.33333   3.33333   0.00000
```

```
link_flows(:,:,5) =

   0.00000       NaN       NaN       NaN       NaN
   2.50000   0.00000       NaN       NaN       NaN
   5.00000       NaN   0.00000       NaN       NaN
   2.50000       NaN       NaN   0.00000       NaN
       NaN   2.50000   5.00000   2.50000   0.00000

link_flows(:,:,6) =

     0   NaN   NaN   NaN   NaN
     4     0   NaN   NaN   NaN
     4   NaN     0   NaN   NaN
     2   NaN   NaN     0   NaN
   NaN     4     4     2     0

link_flows(:,:,7) =

   0.00000       NaN       NaN       NaN       NaN
   3.33333   0.00000       NaN       NaN       NaN
   5.00000       NaN   0.00000       NaN       NaN
   1.66667       NaN       NaN   0.00000       NaN
       NaN   3.33333   5.00000   1.66667   0.00000
```

and the link travel times are as follows:

```
link_times(:,:,1) =

    0.00000        Inf        Inf        Inf        Inf
    5.00000    0.00000        Inf        Inf        Inf
   10.00000        Inf    0.00000        Inf        Inf
   12.50000        Inf        Inf    0.00000        Inf
        Inf    5.00000   10.00000   12.50000    0.00000

link_times(:,:,2) =

     0.00000         Inf         Inf         Inf         Inf
   473.75000     0.00000         Inf         Inf         Inf
    10.00000         Inf     0.00000         Inf         Inf
    12.50000         Inf         Inf     0.00000         Inf
         Inf   473.75000    10.00000    12.50000     0.00000

link_times(:,:,3) =

    0.00000        Inf        Inf        Inf        Inf
   34.29688    0.00000        Inf        Inf        Inf
   13.66211        Inf    0.00000        Inf        Inf
   12.50000        Inf        Inf    0.00000        Inf
        Inf   34.29688   13.66211   12.50000    0.00000

link_times(:,:,4) =

    0.00000        Inf        Inf        Inf        Inf
   10.78704    0.00000        Inf        Inf        Inf
   10.72338        Inf    0.00000        Inf        Inf
   15.35780        Inf        Inf    0.00000        Inf
        Inf   10.78704   10.72338   15.35780    0.00000
```

```
link_times(:,:,5) =

    0.00000        Inf        Inf        Inf        Inf
    6.83105    0.00000        Inf        Inf        Inf
   13.66211        Inf    0.00000        Inf        Inf
   13.40422        Inf        Inf    0.00000        Inf
        Inf    6.83105   13.66211   13.40422    0.00000

link_times(:,:,6) =

    0.00000        Inf        Inf        Inf        Inf
   17.00000    0.00000        Inf        Inf        Inf
   11.50000        Inf    0.00000        Inf        Inf
   12.87037        Inf        Inf    0.00000        Inf
        Inf   17.00000   11.50000   12.87037    0.00000

link_times(:,:,7) =

    0.00000        Inf        Inf        Inf        Inf
   10.78704    0.00000        Inf        Inf        Inf
   13.66211        Inf    0.00000        Inf        Inf
   12.67861        Inf        Inf    0.00000        Inf
        Inf   10.78704   13.66211   12.67861    0.00000
```

As can be seen, the algorithm does not converge particularly rapidly; it does converge, however—although not necessarily monotonically. For this, we turn to the Frank-Wolfe algorithm.

7 User Equilibrium via Frank-Wolfe Algorithm

When deployed in the user equilibrium problem, the Frank-Wolfe algorithm selects the convex combination weighting parameter $\phi \in [0, 1]$ such that the user equilibrium objective function

$$\sum_a \int_0^x t_a(\omega) d\omega \tag{8}$$

is minimized. This is generally done via a line search on the parameter domain; this is generally not too computationally difficult, seeing as the domain is simply $[0, 1]$. The following function performs the UE assignment via Frank-Wolfe:

```
function [link_flows, link_times] = uefw(ladder, bidirectional, od_demand, iters)

[skeleton, capacities] = build_skeleton(ladder, bidirectional);

n = length(skeleton);
link_flows = zeros(n, n, iters + 1);

aux_flows = zeros(n, n, iters);

link_times = zeros(n, n, iters + 1);
link_times(:, :, 1) = skeleton;

for i = 1:iters
  aux_flows(:, :, i) = aon(link_times(:, :, i), od_demand);

  if (i == 1)
    phi = 1;
    ueof_val = ueof(phi, squeeze(link_flows(:, :, i)), ...
      squeeze(aux_flows(:, :, i)), skeleton, capacities)
  else
    [phi, ueof_val] = ueof_line_search(squeeze(link_flows(:, :, i)), ...
      squeeze(aux_flows(:, :, i)), skeleton, capacities)
  end

  link_flows(:, :, i + 1) = link_flows(:, :, i) * (1 - phi) + ...
    aux_flows(:, :, i) * phi;

  link_times(:, :, i + 1) = ...
    update_link_times(skeleton, capacities, squeeze(link_flows(:, :, i + 1)));
end
```

As can be seen, the function is identical to the MSA function with the exception of the assignment of ϕ, which is handled by the block beginning on line 10. On the first iteration, ϕ is set to 1 (lines 10-11); otherwise, the function `ueof_line_search` is called to perform the line search for the optimal value of ϕ (line 14). This function is as follows:

```
function [phi, ueof_val] = ...
  ueof_line_search(link_flows, aux_flows, skeleton, capacities)

interval = 0.1;
tol = 1e-3;

a = 0;
b = 1;

while (interval > tol)
  x = a:interval:(b - interval);
  ueof_vals = zeros(1, 10);
  for i = 1:10
    ueof_vals(i) = ...
      ueof(x(i) + interval / 2, link_flows, aux_flows, skeleton, capacities);
  end
  [min_ueof, i] = min(ueof_vals);
  a = x(i);
  b = x(i) + interval;
  interval = interval / 10;
end

phi = (a + b) / 2;

ueof_val = ueof(phi, link_flows, aux_flows, skeleton, capacities);
```

This function finds the value of ϕ in $[0, 1]$ which yields the minimum value of the UE objective function ('UEOF') in a simple (if somewhat brutal and inelegant) fashion. It begins by dividing the domain into 10 intervals of equal size (line 7; note that a line broken in two by the '...' notation is to be understood as a single line). It then samples the UEOF at the midpoint of each interval (lines 9-11) and determines the interval whose midpoint has a minimum value (line 12). Finally, it sets the interval as the new domain (lines 13-14) and reduces the interval size by a factor of 10 (line 15). This process is repeated until the interval size is equal to the tolerance (the control condition is on line 6), at which point the value of ϕ and the corresponding value of the UEOF are returned (lines 17-18).

The `ueof_line_search` function calls `ueof`, which returns the value of the UEOF given a value for ϕ, a set of link flows, a set of auxiliary flows, and the network information (the base travel times—the '`skeleton`'—and the link `capacities`). This function is as follows:

```
function ueof_val = ueof(phi, link_flows, aux_flows, skeleton, capacities)

ueof_val = 0;
n = length(link_flows);

for i = 1:n
  for j = 1:n
    if (capacities(i, j) > 0)
      updated_flow = link_flows(i, j) * (1 - phi) + aux_flows(i, j) * phi;
      ueof_val = ueof_val + ...
        quadl('bpr', 0, updated_flow, [], [], skeleton(i, j), capacities(i, j));
    end
  end
end
```

`ueof` calls `quadl`, a built-in Octave function which performs numerical integration of a function of a single variable over a finite domain. Inside the `quadl` call is a reference to the function `bpr`, which is a very simple function which returns a link travel time `t` given a volume `x`, a base travel time `t0`, and a link 'capacity' (in the BPR LPF sense) `c`. The function looks like this:

```
function t = bpr(x, t0, c)

alpha = 0.15;
beta = 4;

t = t0 .* (1 + (alpha .*  (x ./ c) .^ beta));
```

Note that the matrix element-by-element operators for multiplication, division, and exponentiation are used; this is for compatibility with `quadl`, which passes vectors to the function whose integral is to be computed.[2,3]

[2]Type `help quadl` at the Octave prompt for more information about `quadl`, or `type quadl` to see the entire function.

[3]MATLAB also has a `quadl` function, which is nominally identical to Octave's (or rather, Octave's was designed to mimic MATLAB's). As of this writing however it is not known whether MATLAB `quadl` will pass the last two arguments to `bpr`, discard them, or break. In order for `ueof` to work properly, the last two parameters of the `quadl` call must be passed to `bpr`.

The call `[link_flows, link_times] = uefw(ladder, 0, od_demand, 5)`, with `ladder` and `od_demand` as above, yields the following link flows. (Recall that the entries with third index k correspond to the values at iteration $k-1$; entries with $k=1$ are zero, so are omitted.)

```
link_flows(:,:,2) =

     0   NaN   NaN   NaN   NaN
    10     0   NaN   NaN   NaN
     0   NaN     0   NaN   NaN
     0   NaN   NaN     0   NaN
   NaN    10     0     0     0

link_flows(:,:,3) =

   0.00000       NaN       NaN       NaN       NaN
   4.05000   0.00000       NaN       NaN       NaN
   5.95000       NaN   0.00000       NaN       NaN
   0.00000       NaN       NaN   0.00000       NaN
       NaN   4.05000   5.95000   0.00000   0.00000

link_flows(:,:,4) =

   0.00000       NaN       NaN       NaN       NaN
   3.38175   0.00000       NaN       NaN       NaN
   4.96825       NaN   0.00000       NaN       NaN
   1.65000       NaN       NaN   0.00000       NaN
       NaN   3.38175   4.96825   1.65000   0.00000

link_flows(:,:,5) =

   0.00000       NaN       NaN       NaN       NaN
   3.61339   0.00000       NaN       NaN       NaN
   4.79436       NaN   0.00000       NaN       NaN
   1.59225       NaN       NaN   0.00000       NaN
       NaN   3.61339   4.79436   1.59225   0.00000

link_flows(:,:,6) =

   0.00000       NaN       NaN       NaN       NaN
   3.55919   0.00000       NaN       NaN       NaN
   4.72245       NaN   0.00000       NaN       NaN
   1.71837       NaN       NaN   0.00000       NaN
       NaN   3.55919   4.72245   1.71837   0.00000
```

and the link travel times are as follows (the 'zeroth iteration' contains the base link travel times—the '`skeleton`'—and is omitted here):

```
link_times(:,:,2) =

     0.00000         Inf         Inf         Inf         Inf
   473.75000     0.00000         Inf         Inf         Inf
    10.00000         Inf     0.00000         Inf         Inf
    12.50000         Inf         Inf     0.00000         Inf
         Inf   473.75000    10.00000    12.50000     0.00000

link_times(:,:,3) =

    0.00000        Inf        Inf        Inf        Inf
   17.61134    0.00000        Inf        Inf        Inf
   17.34377        Inf    0.00000        Inf        Inf
   12.50000        Inf        Inf    0.00000        Inf
        Inf   17.61134   17.34377   12.50000    0.00000

link_times(:,:,4) =

    0.00000        Inf        Inf        Inf        Inf
   11.13066    0.00000        Inf        Inf        Inf
   13.56997        Inf    0.00000        Inf        Inf
   12.67157        Inf        Inf    0.00000        Inf
        Inf   11.13066   13.56997   12.67157    0.00000

link_times(:,:,5) =

    0.00000        Inf        Inf        Inf        Inf
   12.99098    0.00000        Inf        Inf        Inf
   13.09581        Inf    0.00000        Inf        Inf
   12.64879        Inf        Inf    0.00000        Inf
        Inf   12.99098   13.09581   12.64879    0.00000

link_times(:,:,6) =

    0.00000        Inf        Inf        Inf        Inf
   12.52220    0.00000        Inf        Inf        Inf
   12.91420        Inf    0.00000        Inf        Inf
   12.70183        Inf        Inf    0.00000        Inf
        Inf   12.52220   12.91420   12.70183    0.00000
```

Iteration-specific values of ϕ and corresponding values of the UEOF are not returned in a separate variable by the current implementation, but it can be easily modified if desired. At present, they are displayed when the function is called from the Octave prompt. The values for the run documented above are as follows:

```
iteration     phi       ueof_val
1             1         1975
2             0.595     197.41
3             0.165     189.93
4             0.035     189.41
5             0.015     189.35
```

As expected, the algorithm converges rapidly and monotonically.

8 User Equilibrium on the Tromaville Network

Thus far a small 'toy' network has been used to demonstrate the implementations of the various traffic assignment techniques. A user equilibrium run via Frank-Wolfe with 5 iterations on the 'Tromaville' network is presented below.

The 'augmented ladder' representation of the network and O-D demand are loaded as follows:

```
ladder = [
1 6 1 900;
2 9 1 900;
3 12 1 900;
4 11 1 900;
5 15 1 900;
6 7 2 200;
6 9 4 200;
6 12 5 200;
7 8 5 200;
8 12 2 200;
9 10 2 200;
9 11 5 200;
9 12 7 200;
10 11 1 200;
10 12 2 200;
11 14 4 200;
11 16 8 200;
12 13 2 200;
13 16 4 200;
14 15 2 200;
15 16 2 200];

od_demand = [
70 40 10 35 10;
50 100 10 75 20;
140 110 115 260 80;
10 25 10 45 10;
5 5 5 10 10];
```

The following call is made to `uefw`:

```
[link_flows, link_times] = uefw(ladder, 1, od_demand, 5)
```

and the following link flows, link times, and [ϕ, UEOF] pairs are generated:

```
link_flows(:,:,2) =

     0   NaN   NaN   NaN   NaN   205   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
   NaN     0   NaN   NaN   NaN   NaN   NaN   NaN   180   NaN   NaN   NaN   NaN   NaN   NaN   NaN
   NaN   NaN     0   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN    35   NaN   NaN   NaN   NaN
   NaN   NaN   NaN     0   NaN   NaN   NaN   NaN   NaN   NaN   380   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN     0   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   120   NaN
    95   NaN   NaN   NaN   NaN     0     0   NaN    65   NaN   NaN   140   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN     0     0     0   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN   NaN     0     0   NaN   NaN   NaN     0   NaN   NaN   NaN   NaN
   NaN   155   NaN   NaN   NaN    75   NaN   NaN     0   155     0     0   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   140     0    55   370   NaN   NaN   NaN   NaN
   NaN   NaN   NaN    55   NaN   NaN   NaN   NaN     0   390     0   NaN   NaN    20   NaN     0
   NaN   NaN   590   NaN   NaN    20   NaN     0     0    20   NaN     0     5   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN    90     0   NaN   NaN     5
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN    30   NaN   NaN     0    20   NaN
   NaN   NaN   NaN   NaN    25   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN    30     0    90
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN     0   NaN    90   NaN     5     0

link_flows(:,:,3) =

 Columns 1 through 8:

     0.00000         NaN         NaN         NaN         NaN   205.00000         NaN         NaN
         NaN     0.00000         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN     0.00000         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN     0.00000         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN     0.00000         NaN         NaN         NaN
    95.00000         NaN         NaN         NaN         NaN     0.00000     0.00000         NaN
         NaN         NaN         NaN         NaN         NaN     0.00000     0.00000     0.00000
         NaN         NaN         NaN         NaN         NaN         NaN     0.00000     0.00000
         NaN   155.00000         NaN         NaN         NaN    75.00000         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN    55.00000         NaN         NaN         NaN         NaN
         NaN         NaN   590.00000         NaN         NaN    20.00000         NaN     0.00000
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN    25.00000         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN

 Columns 9 through 16:

         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
   180.00000         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN    35.00000         NaN         NaN         NaN         NaN
         NaN         NaN   380.00000         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN   120.00000         NaN
    65.00000         NaN         NaN   140.00000         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN     0.00000         NaN         NaN         NaN         NaN
     0.00000   142.35000     0.00000    12.65000         NaN         NaN         NaN         NaN
   125.05000     0.00000    55.00000   357.35000         NaN         NaN         NaN         NaN
    14.95000   375.05000     0.00000         NaN         NaN    20.00000         NaN     0.00000
     0.00000    20.00000         NaN     0.00000     5.00000         NaN         NaN         NaN
         NaN         NaN         NaN    90.00000     0.00000         NaN         NaN     5.00000
         NaN         NaN    30.00000         NaN         NaN     0.00000    20.00000         NaN
         NaN         NaN         NaN         NaN         NaN    30.00000     0.00000    90.00000
         NaN         NaN     0.00000         NaN    90.00000         NaN     5.00000     0.00000
```

```
link_flows(:,:,4) =

 Columns 1 through 8:

     0.00000         NaN         NaN         NaN         NaN   205.00000         NaN         NaN
         NaN     0.00000         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN     0.00000         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN     0.00000         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN     0.00000         NaN         NaN         NaN
    95.00000         NaN         NaN         NaN         NaN     0.00000     0.00000         NaN
         NaN         NaN         NaN         NaN         NaN     0.00000     0.00000     0.00000
         NaN         NaN         NaN         NaN         NaN         NaN     0.00000     0.00000
         NaN   155.00000         NaN         NaN         NaN    75.00000         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN    55.00000         NaN         NaN         NaN         NaN
         NaN         NaN   590.00000         NaN         NaN    20.00000         NaN     0.00000
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN    25.00000         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN

 Columns 9 through 16:

         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
   180.00000         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN    35.00000         NaN         NaN         NaN         NaN
         NaN         NaN   380.00000         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN   120.00000         NaN
    65.00000         NaN         NaN   140.00000         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN     0.00000         NaN         NaN         NaN         NaN
     0.00000   137.96925     0.00000    17.03075         NaN         NaN         NaN         NaN
   125.72275     0.00000    55.00000   352.96925         NaN         NaN         NaN         NaN
    14.27725   375.72275     0.00000         NaN         NaN    20.00000         NaN     0.00000
     0.00000    20.00000         NaN     0.00000     5.00000         NaN         NaN         NaN
         NaN         NaN         NaN    90.00000     0.00000         NaN         NaN     5.00000
         NaN         NaN    30.00000         NaN         NaN     0.00000    20.00000         NaN
         NaN         NaN         NaN         NaN         NaN    30.00000     0.00000    90.00000
         NaN         NaN     0.00000         NaN    90.00000         NaN     5.00000     0.00000
```

```
link_flows(:,:,5) =

 Columns 1 through 8:

     0.00000         NaN         NaN         NaN         NaN   205.00000         NaN         NaN
         NaN     0.00000         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN     0.00000         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN     0.00000         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN     0.00000         NaN         NaN         NaN
    95.00000         NaN         NaN         NaN         NaN     0.00000     0.00000         NaN
         NaN         NaN         NaN         NaN         NaN     0.00000     0.00000     0.00000
         NaN         NaN         NaN         NaN         NaN         NaN     0.00000     0.00000
         NaN   155.00000         NaN         NaN         NaN    75.00000         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN    55.00000         NaN         NaN         NaN         NaN
         NaN         NaN   590.00000         NaN         NaN    20.00000         NaN     0.00000
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN    25.00000         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN

 Columns 9 through 16:

         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
   180.00000         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN    35.00000         NaN         NaN         NaN         NaN
         NaN         NaN   380.00000         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN   120.00000         NaN
    65.00000         NaN         NaN   140.00000         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN     0.00000         NaN         NaN         NaN         NaN
     0.00000   139.75748     0.00000    15.24252         NaN         NaN         NaN         NaN
   127.22186     0.00000    55.00000   354.75748         NaN         NaN         NaN         NaN
    12.77814   377.22186     0.00000         NaN         NaN    20.00000         NaN     0.00000
     0.00000    20.00000         NaN     0.00000     5.00000         NaN         NaN         NaN
         NaN         NaN         NaN    90.00000     0.00000         NaN         NaN     5.00000
         NaN         NaN    30.00000         NaN         NaN     0.00000    20.00000         NaN
         NaN         NaN         NaN         NaN         NaN    30.00000     0.00000    90.00000
         NaN         NaN     0.00000         NaN    90.00000         NaN     5.00000     0.00000
```

```
link_flows(:,:,6) =

 Columns 1 through 8:

     0.00000         NaN         NaN         NaN         NaN   205.00000         NaN         NaN
         NaN     0.00000         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN     0.00000         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN     0.00000         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN     0.00000         NaN         NaN         NaN
    95.00000         NaN         NaN         NaN         NaN     0.00000     0.00000         NaN
         NaN         NaN         NaN         NaN         NaN     0.00000     0.00000     0.00000
         NaN         NaN         NaN         NaN         NaN         NaN     0.00000     0.00000
         NaN   155.00000         NaN         NaN         NaN    75.00000         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN    55.00000         NaN         NaN         NaN         NaN
         NaN         NaN   590.00000         NaN         NaN    20.00000         NaN     0.00000
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN    25.00000         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN

 Columns 9 through 16:

         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
   180.00000         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN    35.00000         NaN         NaN         NaN         NaN
         NaN         NaN   380.00000         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN   120.00000         NaN
    65.00000         NaN         NaN   140.00000         NaN         NaN         NaN         NaN
         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN
         NaN         NaN         NaN     0.00000         NaN         NaN         NaN         NaN
     0.00000   138.33612     0.00000    16.66388         NaN         NaN         NaN         NaN
   127.41353     0.00000    55.00000   353.33612         NaN         NaN         NaN         NaN
    12.58647   377.41353     0.00000         NaN         NaN    20.00000         NaN     0.00000
     0.00000    20.00000         NaN     0.00000     5.00000         NaN         NaN         NaN
         NaN         NaN         NaN    90.00000     0.00000         NaN         NaN     5.00000
         NaN         NaN    30.00000         NaN         NaN     0.00000    20.00000         NaN
         NaN         NaN         NaN         NaN         NaN    30.00000     0.00000    90.00000
         NaN         NaN     0.00000         NaN    90.00000         NaN     5.00000     0.00000
```

```
link_times(:,:,2) =

 Columns 1 through 9:

   0.00000       Inf       Inf       Inf       Inf   1.00040       Inf       Inf       Inf
       Inf   0.00000       Inf       Inf       Inf       Inf       Inf       Inf   1.00024
       Inf       Inf   0.00000       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf   0.00000       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf   0.00000       Inf       Inf       Inf       Inf
   1.00002       Inf       Inf       Inf       Inf   0.00000   2.00000       Inf   4.00669
       Inf       Inf       Inf       Inf       Inf   2.00000   0.00000   5.00000       Inf
       Inf       Inf       Inf       Inf       Inf       Inf   5.00000   0.00000       Inf
       Inf   1.00013       Inf       Inf       Inf   4.01187       Inf       Inf   0.00000
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf   2.07203
       Inf       Inf       Inf   1.00000       Inf       Inf       Inf       Inf   5.00000
       Inf       Inf   1.02770       Inf       Inf   5.00008       Inf   2.00000   7.00000
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf   1.00000       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf

 Columns 10 through 16:

       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf   1.00000       Inf       Inf       Inf       Inf
       Inf   1.00477       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf   1.00005       Inf
       Inf       Inf   5.18007       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf   2.00000       Inf       Inf       Inf       Inf
   2.10823   5.00000   7.00000       Inf       Inf       Inf       Inf
   0.00000   1.00086   5.51405       Inf       Inf       Inf       Inf
   3.16885   0.00000       Inf       Inf   4.00006       Inf   8.00000
   2.00003       Inf   0.00000   2.00000       Inf       Inf       Inf
       Inf       Inf   2.01230   0.00000       Inf       Inf   4.00000
       Inf   4.00030       Inf       Inf   0.00000   2.00003       Inf
       Inf       Inf       Inf       Inf   2.00015   0.00000   2.01230
       Inf   8.00000       Inf   4.02460       Inf   2.00000   0.00000
```

```
link_times(:,:,3) =

 Columns 1 through 9:

   0.00000       Inf       Inf       Inf       Inf   1.00040       Inf       Inf       Inf
       Inf   0.00000       Inf       Inf       Inf       Inf       Inf       Inf   1.00024
       Inf       Inf   0.00000       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf   0.00000       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf   0.00000       Inf       Inf       Inf       Inf
   1.00002       Inf       Inf       Inf       Inf   0.00000   2.00000       Inf   4.00669
       Inf       Inf       Inf       Inf       Inf   2.00000   0.00000   5.00000       Inf
       Inf       Inf       Inf       Inf       Inf       Inf   5.00000   0.00000       Inf
       Inf   1.00013       Inf       Inf       Inf   4.01187       Inf       Inf   0.00000
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf   2.04585
       Inf       Inf       Inf   1.00000       Inf       Inf       Inf       Inf   5.00002
       Inf       Inf   1.02770       Inf       Inf   5.00008       Inf   2.00000   7.00000
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf   1.00000       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf

 Columns 10 through 16:

       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf   1.00000       Inf       Inf       Inf       Inf
       Inf   1.00477       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf   1.00005       Inf
       Inf       Inf   5.18007       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf   2.00000       Inf       Inf       Inf       Inf
   2.07699   5.00000   7.00002       Inf       Inf       Inf       Inf
   0.00000   1.00086   5.05757       Inf       Inf       Inf       Inf
   2.85493   0.00000       Inf       Inf   4.00006       Inf   8.00000
   2.00003       Inf   0.00000   2.00000       Inf       Inf       Inf
       Inf       Inf   2.01230   0.00000       Inf       Inf   4.00000
       Inf   4.00030       Inf       Inf   0.00000   2.00003       Inf
       Inf       Inf       Inf       Inf   2.00015   0.00000   2.01230
       Inf   8.00000       Inf   4.02460       Inf   2.00000   0.00000
```

```
link_times(:,:,4) =

 Columns 1 through 9:

   0.00000       Inf       Inf       Inf       Inf   1.00040       Inf       Inf       Inf
       Inf   0.00000       Inf       Inf       Inf       Inf       Inf       Inf   1.00024
       Inf       Inf   0.00000       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf   0.00000       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf   0.00000       Inf       Inf       Inf       Inf
   1.00002       Inf       Inf       Inf       Inf   0.00000   2.00000       Inf   4.00669
       Inf       Inf       Inf       Inf       Inf   2.00000   0.00000   5.00000       Inf
       Inf       Inf       Inf       Inf       Inf       Inf   5.00000   0.00000       Inf
       Inf   1.00013       Inf       Inf       Inf   4.01187       Inf       Inf   0.00000
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf   2.04684
       Inf       Inf       Inf   1.00000       Inf       Inf       Inf       Inf   5.00002
       Inf       Inf   1.02770       Inf       Inf   5.00008       Inf   2.00000   7.00000
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf   1.00000       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf

 Columns 10 through 16:

       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf   1.00000       Inf       Inf       Inf       Inf
       Inf   1.00477       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf   1.00005       Inf
       Inf       Inf   5.18007       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf   2.00000       Inf       Inf       Inf       Inf
   2.06794   5.00000   7.00006       Inf       Inf       Inf       Inf
   0.00000   1.00086   4.91037       Inf       Inf       Inf       Inf
   2.86828   0.00000       Inf       Inf   4.00006       Inf   8.00000
   2.00003       Inf   0.00000   2.00000       Inf       Inf       Inf
       Inf       Inf   2.01230   0.00000       Inf       Inf   4.00000
       Inf   4.00030       Inf       Inf   0.00000   2.00003       Inf
       Inf       Inf       Inf       Inf   2.00015   0.00000   2.01230
       Inf   8.00000       Inf   4.02460       Inf   2.00000   0.00000
```

```
link_times(:,:,5) =

 Columns 1 through 9:

   0.00000       Inf       Inf       Inf       Inf   1.00040       Inf       Inf       Inf
       Inf   0.00000       Inf       Inf       Inf       Inf       Inf       Inf   1.00024
       Inf       Inf   0.00000       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf   0.00000       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf   0.00000       Inf       Inf       Inf       Inf
   1.00002       Inf       Inf       Inf       Inf   0.00000   2.00000       Inf   4.00669
       Inf       Inf       Inf       Inf       Inf   2.00000   0.00000   5.00000       Inf
       Inf       Inf       Inf       Inf       Inf       Inf   5.00000   0.00000       Inf
       Inf   1.00013       Inf       Inf       Inf   4.01187       Inf       Inf   0.00000
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf   2.04912
       Inf       Inf       Inf   1.00000       Inf       Inf       Inf       Inf   5.00001
       Inf       Inf   1.02770       Inf       Inf   5.00008       Inf   2.00000   7.00000
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf   1.00000       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf

 Columns 10 through 16:

       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf   1.00000       Inf       Inf       Inf       Inf
       Inf   1.00477       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf   1.00005       Inf
       Inf       Inf   5.18007       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf   2.00000       Inf       Inf       Inf       Inf
   2.07153   5.00000   7.00004       Inf       Inf       Inf       Inf
   0.00000   1.00086   4.96980       Inf       Inf       Inf       Inf
   2.89827   0.00000       Inf       Inf   4.00006       Inf   8.00000
   2.00003       Inf   0.00000   2.00000       Inf       Inf       Inf
       Inf       Inf   2.01230   0.00000       Inf       Inf   4.00000
       Inf   4.00030       Inf       Inf   0.00000   2.00003       Inf
       Inf       Inf       Inf       Inf   2.00015   0.00000   2.01230
       Inf   8.00000       Inf   4.02460       Inf   2.00000   0.00000
```

```
link_times(:,:,6) =

 Columns 1 through 9:

   0.00000       Inf       Inf       Inf       Inf   1.00040       Inf       Inf       Inf
       Inf   0.00000       Inf       Inf       Inf       Inf       Inf       Inf   1.00024
       Inf       Inf   0.00000       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf   0.00000       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf   0.00000       Inf       Inf       Inf       Inf
   1.00002       Inf       Inf       Inf       Inf   0.00000   2.00000       Inf   4.00669
       Inf       Inf       Inf       Inf       Inf   2.00000   0.00000   5.00000       Inf
       Inf       Inf       Inf       Inf       Inf       Inf   5.00000   0.00000       Inf
       Inf   1.00013       Inf       Inf       Inf   4.01187       Inf       Inf   0.00000
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf   2.04942
       Inf       Inf       Inf   1.00000       Inf       Inf       Inf       Inf   5.00001
       Inf       Inf   1.02770       Inf       Inf   5.00008       Inf   2.00000   7.00000
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf   1.00000       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf       Inf

 Columns 10 through 16:

       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf   1.00000       Inf       Inf       Inf       Inf
       Inf   1.00477       Inf       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf   1.00005       Inf
       Inf       Inf   5.18007       Inf       Inf       Inf       Inf
       Inf       Inf       Inf       Inf       Inf       Inf       Inf
       Inf       Inf   2.00000       Inf       Inf       Inf       Inf
   2.06867   5.00000   7.00005       Inf       Inf       Inf       Inf
   0.00000   1.00086   4.92249       Inf       Inf       Inf       Inf
   2.90213   0.00000       Inf       Inf   4.00006       Inf   8.00000
   2.00003       Inf   0.00000   2.00000       Inf       Inf       Inf
       Inf       Inf   2.01230   0.00000       Inf       Inf   4.00000
       Inf   4.00030       Inf       Inf   0.00000   2.00003       Inf
       Inf       Inf       Inf       Inf   2.00015   0.00000   2.01230
       Inf   8.00000       Inf   4.02460       Inf   2.00000   0.00000
```

iteration	phi	ueof_val
1	1	6519.4
2	0.115	6513.7
3	0.045	6513.4
4	0.105	6513.3
5	0.015	6513.3

The algorithm seems to have converged shockingly rapidly, although this (along with the odd behavior of ϕ) may be cause for some suspicion. What is happening? Is `ueof_line_search` finding a local minimum of the UEOF but not a global one? (Is `ueof_line_search` even suitable for such large networks?) Network visualization tools—not to mention some rigorous analysis of the 'infrastructural' algorithms *as implemented*—would be of great value here; sadly, both are beyond the scope of this paper, which has sought—it is hoped with *some* success—to present some code which points toward a robust and fully general Octave/MATLAB implementation of the suite of traffic assignment algorithms.