Supplemental Information

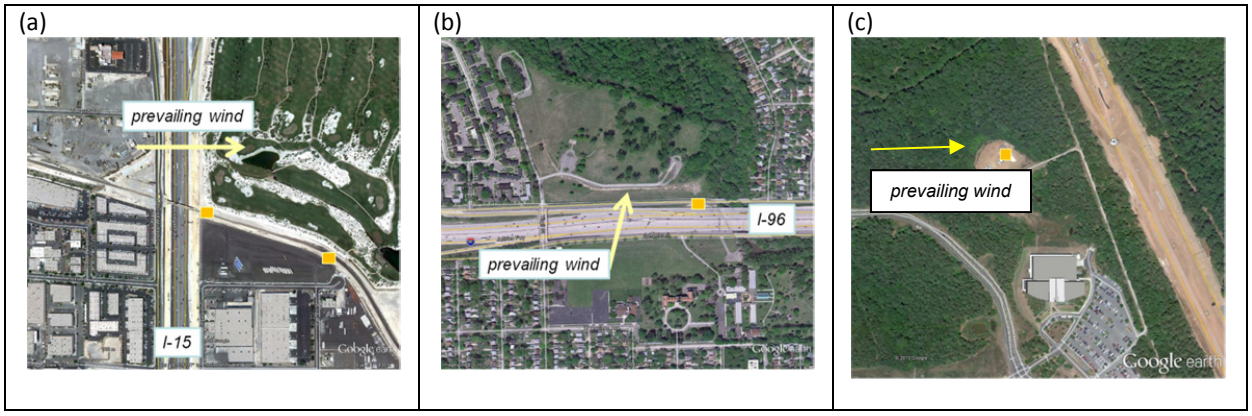

Figure S1: Location of field monitoring sites, LV-road and LV-300m (a), DT-road (b), and RTP-BGD (c). The orange squares note the locations of the UFP monitoring stations. Note that a roadway was under construction approximately 150 m from the RTP-BGD monitoring station at the time of sampling (c).

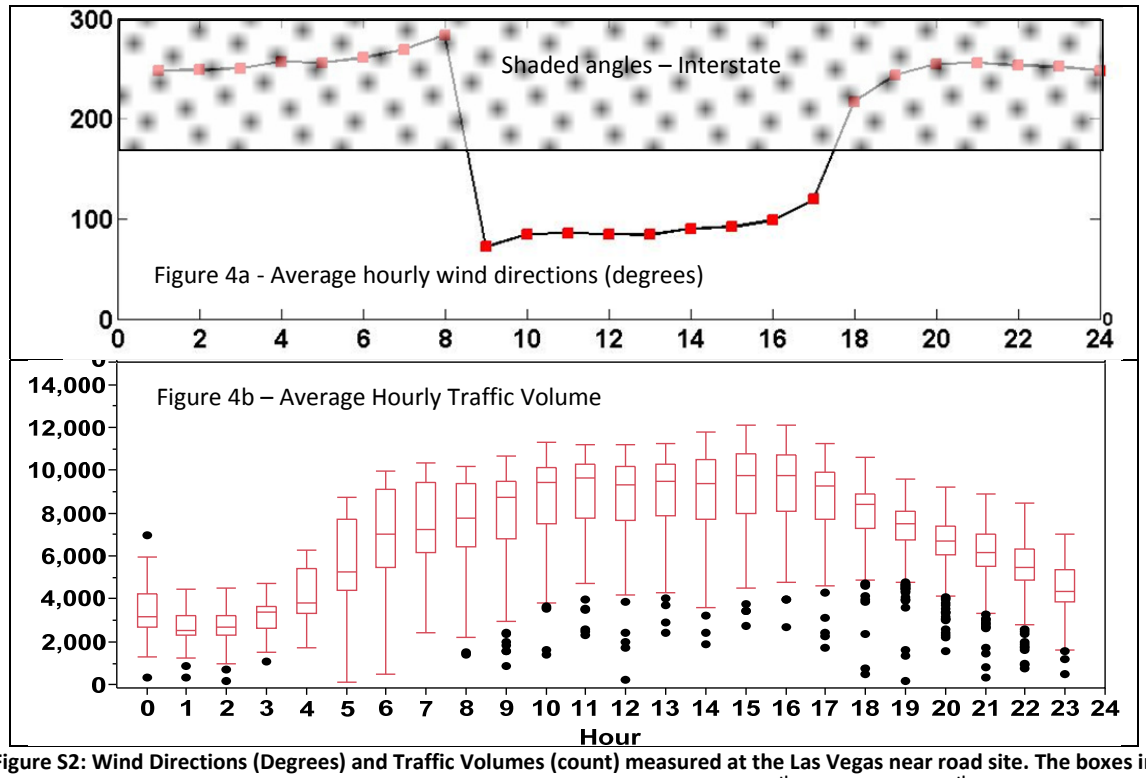

Figure S2: Wind Directions (Degrees) and Traffic Volumes (count) measured at the Las Vegas near road site. The boxes in Figure 4b represents the inter-quartile range while the whiskers represent the 5th (bottom) and 95th (top) percentiles.

CADETS MATLAB Code

```
%  CADETS code
% This code expects the user to provide several pieces of information:
% 1. data: the measurement data (e.g., a single column of values)
% 2. timeint: the time base of the data (e.g., 15 min)
% 3. butterval: the user provided cut-off frequency.
% 4. butterorder: this value sets the order of the lowpass digital Butterworh filter
% 5. convert2hrs: value to convert timeint to hours (e.g., 15 min / 1440
% min/hr)
% 6. peakwindow: value to set length of the window for finding peaks

%user provided information
%load data
%data is a single column of measurement values provided by the user
timeint = 15; %example provided setting the data time interval at 15 min
butterval = 0.023; % Vedantham et al. (2014) utilized value of 0.023.
butterorder = 10; % Vedantham et al. (2014) utilized value of 10.
convert2hrs = 1440; %converts minutes to days
peakwindow = 50;  % 12 hour fundamental frequency for a 15-minute data translates to
~50 (12
% times 4) as the minpeakdistance. Otherwise, way too many peaks and valley points are
found.

%code begins here
NFFT = 2^nextpow2(length(data))/2;  %sets the number of frequencies to be tested in
fft
freqs = [1:NFFT]'/NFFT;  %normalized to give array of frequencies
periods = 1./freqs';
days = periods*timeint/convert2hrs; % division to convert timebase to hours
data(isnan(data)) = []; %remove any rows with missing values
%%
[b,a] = butter(butterorder,butterval); %10th order butterworth filter with cut-off
frequency of 0.023 (ref. Vedantham et al.)
output = filtfilt(b,a,data); %application of butterworth filter to data set
clear a b
%%
datafft = fft(data,NFFT); %Fast Fourier transfor of the raw data
outputFFT = fft(output,NFFT); %Fast Fourier transform of the data passed through the
butterworth filter
theoreticalLocal = data-output; %subtraction of the low frequency signal from the raw
data
theoLocalFFT = fft(theoreticalLocal,NFFT); %Fast Fourier transform of the estimated
"local" timeseries

clear freqs periods
[valleys,valleyLocs] = findpeaks(-output,'minpeakdistance',peakwindow);  %find local
minima in set window length, using findpeaks command on inversion of output variable
[peaks,peakLocs] = findpeaks(output,'minpeakdistance',peakwindow);  %find local maxima
in set window length

%% Now, using the peak and the valley points, the slowly varying part is reconstructed
extValleyLocs = union(union(1,valleyLocs),length(data)); %appends indices 1 and end of
data to valleyLocs
notLocal = NaN(size(output)); %create notLocal variable, representing slowly varying
timeseries component
exceeds = 0;
needPatchup = 0;
prevRange = 1;
prevMaxLoc = peakLocs(1);
prevPatch = [];
for i = 2:length(extValleyLocs)
```

```
    currRange = extValleyLocs(i-1):extValleyLocs(i);  %sets window to span two local
minima
    maxLoc = currRange(find(ismember(currRange,peakLocs)));  %finds local maxima
indices within the set range
    if isempty(maxLoc)  %if no local maxima is within window, sets the max value as
the highest value within the range
        [maxVal,loc] = max(output(currRange));
        maxLoc = currRange(loc(1));
    else
        maxLoc = maxLoc(1); %otherwise, selects first occurrence of a local maxima in
the window
    end
    spanVal = 5;
    while exceeds < 2
        thisRange = max(maxLoc-
spanVal,currRange(1)):min(maxLoc+spanVal,currRange(end));  %creates a window spanning
spanVal indices +/- the location of the maxLoc
        if length(find(ismember(thisRange,currRange))) < length(currRange)
            thisPatch = NaN(size(output));
            thisPatch(thisRange) = output(thisRange);  %apply output values for the
local maxima window
            thisPatch = thisPatch - nanmin(thisPatch);
            exceeds = length(find(thisPatch(currRange)>...
                data(currRange)))*100/length(currRange);  %compares local output
maxima against original raw data
            spanVal = spanVal + 10;  %expands span window and repeats
        else
            break;
        end
    end
    if length(find(ismember(thisRange,currRange))) == length(currRange)
        maxValDiff = output(maxLoc) - max(thisPatch); %compares output local maximum
versus maximum estimated for window
        factorVal = 0.1;
        thisPatch = NaN(size(output));
        thisPatch(thisRange)  = output(thisRange);
        exceeds = length(find(thisPatch(currRange)>...
            data(currRange)))*100/length(currRange);
        if exceeds <= 2
            break
        else
            while exceeds > 2 %2% is the user-chosen tolerance (step 5 of the paper)
                thisPatch = thisPatch - factorVal*maxValDiff;   %decreases estimated
value
                exceeds = length(find(thisPatch(currRange)>...
                    data(currRange)))*100/length(currRange);
                factorVal = factorVal + 0.05;                   %increases factorVal
slightly and repeats
            end
            thisPatch(thisPatch < 0) = NaN;
        end
    end
    if prevRange(end) == thisRange(1) % Arrive at the patch up if the previous patch
was the entire intrval
        cutoffLoc = find(prevPatch(prevMaxLoc:prevRange(end))<thisPatch(1),1,'first');
        if isempty(cutoffLoc)
            shortStretch = min(maxLoc - currRange(1),currRange(end)-maxLoc);
            cutoff = floor(0.3*shortStretch);
            thisPatch([currRange(1):currRange(1)+cutoff ...
                currRange(end)-cutoff:currRange(end)]) = NaN;

        end
        nonLocal(cutoffLoc:prevRange(end)) = NaN;
    end
```

```
    nonLocal(currRange) = thisPatch(currRange);
    lastNonzeroVal = find(~isnan(nonLocal(prevRange)),1,'last');
    firstNonzeroVal = find(~isnan(nonLocal(thisRange)),1,'first');
    if nonLocal(prevRange(lastNonzeroVal)) == 0 & ...
            nonLocal(thisRange(firstNonzeroVal)) == 0
        nonLocal(prevRange(lastNonzeroVal):thisRange(firstNonzeroVal)) = 0;
    end

    prevRange = currRange;
    prevPatch = thisPatch;
    prevMaxLoc = maxLoc;
    exceeds = 0;
end

missingVals = find(isnan(nonLocal));
nonMissingVals = setdiff(1:length(nonLocal),missingVals);
%the next line interpolates the missing spots using cubic smoothing splines
connections = csaps(nonMissingVals,nonLocal(nonMissingVals),1,missingVals);
completeNonLocal = nonLocal';
completeNonLocal(missingVals) = connections;
completeNonLocal(completeNonLocal < 0) = 0; %set negative values to zero
completeNonLocalFFT = fft(completeNonLocal,NFFT);
completeLocal = data-completeNonLocal(1:length(data));
completeLocal(completeLocal<0) = 0; %set negative values to zero
completeLocalFFT = fft(completeLocal,NFFT);
data(data == 0) = NaN;
nanmean(completeNonLocal(1:length(data))./data)
%remove temporary variables
clear NFFT exceeds connections prevPatch prevMaxLoc prevRange lastNonzeroVal
firstNonzeroVal i needPatchup maxLoc maxVal
clear maxValDif peaks peakLocs thisPatch thisRange timeint valleys currRange
convert2hrs loc spanVal shortStretch cutoff
clear butterorder ans utterval allTogether cuttoff cutoffLoc maxValDiff factorVal
extValleyLocs peakwindow valleyLocs butterval
return

%%
figure
dataPlot = plot(data,'DisplayName','data','YDataSource','data');figure(gcf)
axis tight
ylim([0 38000])
xlim([12000 19000])
hold on
% thisPlot = plot(output,'c','linewidth',2);
nonLocal = plot(completeNonLocal,'r');

%%
figure;
fftPlot = plot(days,abs(datafft).^2);
axis tight
currAxes = gca;
set(currAxes,'xscale','log');
xlims = [0.3 90];
xlim(xlims)
ylim([0 2.5e15]);
ax2 = axes('Position',get(currAxes,'Position'));
localFFT = plot(ax2,days,abs(theoLocalFFT).^2,'r','linewidth',3);
set(ax2,'xscale','log','yaxislocation','right','box','off','xlim',xlims,'color','none')
set(currAxes,'box','off');
set(ax2,'box','off');
set(ax2,'color','none','xlim',xlims);
ylim(currAxes,[0 1e15])
ylim(ax2,[0 1e15]);
```

```
legend([fftPlot localFFT],{'Original data FFT','CADETS - Highly Varying FFT'});
%%
figure;
fftPlot = plot(days,abs(datafft).^2);
axis tight
currAxes = gca;
set(currAxes,'xscale','log');
xlims = [0.3 90];
xlim(xlims)
ylim([0 2.5e15]);
ax2 = axes('Position',get(currAxes,'Position'));
decomposedFFT = plot(ax2,days,abs(outputFFT).^2,'r','linewidth',3);
set(ax2,'xscale','log','yaxislocation','right','box','off','xlim',xlims,'color','none')
set(currAxes,'box','off');
set(ax2,'box','off');
set(ax2,'color','none','xlim',xlims);
ylim(currAxes,[0 1e15])
ylim(ax2,[0 1e15]);
legend([fftPlot decomposedFFT],{'Original data FFT','CADETS - Slowly Varying FFT'});
```