

Vectorized Spectra Generation Algorithm in GOBASIC

Luyao Zou

February 27, 2015

1 Summary

As a script language, Matlab is optimized for vector operations, but has poor for-loop performance. This document describes the details of vectorized algorithm of spectra generation in GOBASIC program[1]. Both the codes of the old for-loop algorithm and the current one are listed.

The vectorized algorithm in general can improve the speed of spectra simulation by about an order of magnitude. If you have better ideas to improve the algorithm please feel free to contact me at luyao.zou@emory.edu

2 Old Algorithm: A For-loop Solution

A spectrum for a given fitting component is generated in GOBASIC based on the function

$$\mathbf{y}_{\text{fit}} = \sum_m \gamma_m(\mathbf{x}) \quad (1)$$

where \mathbf{x} is the x-data, γ is the Gaussian profile function, T_b is the brightness temperature, and m stands for a molecular transition listed in the catalog.

In GOBASIC, T_{b_m} is returned by function `maxtempcalc.m` as a vector, `maxtemp`, for a given fitting component. The old program takes this vector, matches it with the catalog files and simulation parameters, and generates a spectrum. The old code is listed:

```

1 function molsim = molsim(maxtemp,obsfreq,molfreq,fwhm_v,shift_v)
2 c = 299792.458;
3 molsim = zeros(length(obsfreq),1);
4 fwhm = fwhm_v*median(obsfreq)/c;
5 shift = shift_v*median(obsfreq)/c;
6 const = fwhm^2/(4*log(2));
7 for n=1:length(maxtemp)
8     line = maxtemp(n,1)*exp(-(obsfreq-molfreq(n,1)+shift).^2/const);
9     molsim = molsim + line;
10 end

```

`molfreq`, taken from the catalog, stores the center frequency of transitions of the given component. `length(molfreq) = length(maxtemp)`. Though very simple, this for-loop solution is quite inefficient, as for each evaluation of a set of parameters during the optimization, thousands of for-loop cycles need to be executed. To be fair, it usually takes hundreds of thousands of function evaluations during one optimization of a broadband observational dataset. So the total numbers of for-loop cycles executed will be huge.

3 Vectorized Algorithm: A Faster Solution

The vectorized algorithm tries to replace the for-loop with a summation of a set of individual vectors, each representing a Gaussian line-shape of one transition. In addition, for each individual Gaussian line-shape function, it is not necessary to calculate the value over the whole GHz-wide observational frequency window. Most of the values will be so close to zero that they can be omitted. Therefore, sparse matrix methods can be used to save computing resources. (In fact, trying to create a full matrix can easily exceed Matlab's upper limit of matrix size.) In the current version of GOBASIC (4.3), a 200 x-data points are used as the window for line-shape generation, defined as variable `range` in `definevars.m`. This range is decent for the CSO dataset, since the frequency step size is 1 MHz and the typical line-width is 10 MHz. Thus, 201 x-data points stand for a 200 MHz frequency window, which is on the scale of $\pm 20\sigma$ wide, sufficient to create an precise Gaussian line profile.

The equation used to calculate the spectral function is also modified, according to the suggestion of an anonymous referee. The new function,

which is described in the paper, is

$$\mathbf{y}_{\text{fit}} = f(\mathbf{x}; N_{\text{T}}, \Delta v, T, v_{\text{off}}) = \sum_n \sum_{\nu_c} \gamma_c(\mathbf{x}) = \sum_n \sum_{\nu_c} \eta_{\text{B}} \frac{h\nu_c}{k} \frac{1 - e^{-\tau_c(\mathbf{x})}}{e^{h\nu_c/kT} - 1} \quad (2)$$

where

$$\eta_{\text{B}} = \frac{\theta_{\text{s}}^2}{\theta_{\text{s}}^2 + \theta_{\text{b}}^2} \quad (3)$$

is the beam filling factor (frequency dependent and calculated on the fly);

$$\tau_c(\mathbf{x}) = \frac{1}{4} \sqrt{\frac{\ln 2}{\pi^3}} \frac{c^2}{\nu_c^2 \Delta \nu} \frac{N_{\text{T}} A_{\text{ul}} g_{\text{u}}}{Q(T)} e^{-E_{\text{u}}/kT} (e^{h\nu_c/kT} - 1) g(\mathbf{x}) \quad (4)$$

is the optical depth of transition c; and

$$g(x) = \exp \left(- \frac{4 \ln 2 (x - \nu_c + \nu_{\text{off}})^2}{\Delta \nu^2} \right) \quad (5)$$

is the Gaussian profile.

Since both the beam filling factor and the linewidth and shift in velocity units are frequency dependent, all these values are calculated on the fly. As a result, all calculations have been combined into `totalsim.m`, whereas `molssim.m` and `maxtempcalc.m` are no longer needed (v4.3 and later).

3.1 Generate Indices

The first thing required by this solution, is to have a index matrix marks out the index range in the x-data to generate the Gaussian. The reason of making this matrix is also to match up the requirements of Matlab sparse matrix operations, which will be discussed in the next section. This index matrix is returned by function `linecenter.m`:

```

1 function linecenter_idx = linecenter(molfreq,obsfreq,range)
2 indices = zeros(1,length(molfreq));
3 for n = 1:length(molfreq)
4     dfreq = abs(obsfreq - molfreq(n));
5     index = find(dfreq == min(dfreq));
6     indices(1,n) = index(1);

```

```

7 end
8 range_min = -idivide(int16(range),2,'ceil');
9 range_max = idivide(int16(range),2,'ceil');
10 range_matrix = repmat((range_min:range_max)',1,length(molfreq));
11 linecenter_idx = repmat(indicies,range_max-range_min+1,1) - double(range_matrix);

```

This function takes three arguments, `molfreq`, `obsfreq`, and `range`. `molfreq` is the molecular transition frequency vector extracted from the catalog.

$$\mathbf{molfreq} = [\nu_1, \nu_2, \nu_3, \dots, \nu_m]^T \quad m \text{ transitions}$$

`obsfreq` is the frequency-x points extracted from the observational data.

$$\mathbf{obsfreq} = [x_1, x_2, x_3, \dots, x_n]^T \quad n \text{ x-data points}$$

Line 2 creates a zero column vector in the same length of `molfreq`.

Line 3–7 is a for-loop, which I did not find a way to get rid of, to return the line center indices in `obsfreq` of all ν_m . This is because the x-vector from observations has some finite channel width (on the scale of 1 MHz in our CSO line surveys). It is likely that the line center ν_m taken from the catalog list does not match the x-data points. But we can find a closest number as the starting point of generating Gaussian windows.

Line 4 calculate the difference between x_i and ν_m , `dfreq`.

Line 5 finds the index of the minimum of `dfreq`, which is the index of the x-data point closest to the frequency center ν_m .

Line 6 stores that index into `indicies`=[$i_1, i_2, i_3, \dots, i_m$].

Line 8–10 creates the Gaussian window cutoff matrix, from $-\text{range}/2$ to $+\text{range}/2$ replicated by `length(molfreq)` times.

$$\mathbf{r} = \mathbf{range_matrix} = \begin{bmatrix} -\text{range}/2 & -\text{range}/2 & \dots & -\text{range}/2 \\ -\text{range}/2 + 1 & -\text{range}/2 + 1 & \dots & -\text{range}/2 + 1 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ \text{range}/2 & \text{range}/2 & \dots & \text{range}/2 \end{bmatrix}$$

Line 11 creates the index matrix for the Gaussian windows, centered at transition frequencies from the catalog. It first replicates the index matrix

to match the dimension of `range_matrix`, and the subtract `range_matrix` from it.

$$\begin{aligned} \mathbf{i} &= \text{repmat}(\text{indicies}, \text{range_max} - \text{range_min} + 1, 1) \\ &= \begin{bmatrix} i_1 & i_2 & \dots & i_m \\ i_1 & i_2 & \dots & i_m \\ \vdots & \vdots & & \vdots \\ i_1 & i_2 & \dots & i_m \end{bmatrix} \\ \text{linecenter_idx} &= \mathbf{i} - \mathbf{r} \end{aligned}$$

Now this `linecenter_idx` is a index matrix which labels the indices in `obsfreq` to generate Gaussian profiles for each molecular transition. This function still uses a for-loop, which may cause thousands of cycles. Nevertheless, once generated, `linecenter_idx` is returned and hold unchanged during the whole fitting process. So the for-loop is only executed once, which is acceptable.

Another line in `analysisscript.m` defines

```
1 chwidth = mean(diff(obsfreq));
```

This line calculate the channel width of an observational data set. It is the average value of the differences between adjacent x-data points. This is build upon the assumption that the frequency data are uniformly sampled.

3.2 Generate Gaussian Profile

The function `totalsim` is used to output a complete spectrum vector for a given molecular component. It takes `linecenter_idx` and other arguments. The code is listed:

```
1 function totalsim = totalsim(molfreq,obsfreq,aco,agu,eup,lc_idx,chwidth,...
2                               sourcesize,dish,lognt,fwhm_v,lntemp,shift_v)
3
4 h = 6.62606957e-34;
5 k = 1.3806488e-23;
6 c = 2.99792458e8;
```

```

7
8 temp = exp(lntemp);
9 nt = 10^(lognt+4);
10 nu = molfreq*1e6;
11 shift = shift_v*molfreq/c*1e3;
12 fwhm = fwhm_v*molfreq/c*1e3;
13 linecenter = molfreq - shift;
14 q = aco*temp^(3/2);
15 range_len = length(lc_idx(:,1));
16 % tau at line center
17 tau_c = (sqrt(log(2)/pi^3)/4).*(c.^2./(nu.^2)).*nt./q.*agu.*...
18         exp(-eup./(k.*temp)).*(exp(h.*nu./(k.*temp))-1);
19 % calculate beam filling factor if source size is specified
20 if sourcesize ~= 0
21     lambda = c./(linecenter*1e6);
22     % beam size at line center
23     beamsize = (sqrt(-8*log(2)*log(0.1))*lambda/dish*180*3600/pi^2);
24     beamfilling = repmat(sourcesize,length(beamsize),1).^2./...
25         (repmat(sourcesize,length(beamsize),1).^2+beamsize.^2);
26 end
27 linecenter = repmat(linecenter',range_len,1);
28 fwhm = repmat(fwhm',range_len,1);
29 shift_idx = repmat(round(shift/chwidth)',range_len,1);
30 jlabel = lc_idx - shift_idx;
31 ilabel = (1:length(molfreq));
32 ilabel = repmat(ilabel,range_len,1);
33 tau = repmat(tau_c',range_len,1)./fwhm.*...
34         exp(-4*log(2).*(obsfreq(jlabel)-linecenter).^2./fwhm.^2);
35 % catch if no beam filling correction is specified
36 if sourcesize == 0
37     b = (h.*nu)./(k.*(exp(h.*nu./(k.*temp))-1));
38 else
39     b = (h.*nu)./(k.*(exp(h.*nu./(k.*temp))-1)).*beamfilling;
40 end
41 svalue = repmat(b',range_len,1).*(1-exp(-tau));
42 lines = sparse(reshape(ilabel,1,[]), reshape(jlabel,1,[]),...
43         reshape(svalue,1,[]), length(molfreq),length(obsfreq));
44 totalsim = full(sum(lines,1))';

```

First 15 lines define constant and perform on-the-fly conversion of line centers and shifts from velocity unit into frequency unit.

Line 17 calculate the optical depth vector at the transition center, without the Gaussian profile

Line 19–26 constructs the beam filling factor matrix. Notice that here the wavelength of the light is converted from the transition center corrected by the velocity shift.

Line 27–28 generates line center and line width matrices, `range_len`×`m`.

Line 29–30 generates the shifted line-center index matrix, which has the same number of rows with `molfreq`, and same number of columns with `lc_idx` (`range_len`). The `shift_idx` first calculates how much index numbers needs to be shifted to count for the shift of line centers. This is a sufficiently close estimation from the ratio of shifted frequency and channel width of the observational data. `chwidth` is pre-calculated in `analysisscript.n` to be `chwidth = mean(diff(obsfreq))`. The the index shift is applied to `lc_idx` to produce `jlabel`, which is the used by the following codes to generate the Gaussian line-profiles.

Line 31–32 generates another index matrix, called `ilabel`. This is simply a `range_len`×`m` matrix, each column of which is number 1 through `range_len`.

Line 33–34 replicates the optical depth vector `tau_c`, and multiplies it with the Gaussian profile to generate a τ matrix at each x-data points, of the same dimensions of other index matrices.

Line 35–40 furthermore corrects for beam filling effect, if necessary.

$$\begin{aligned}
\text{linecenter} = \boldsymbol{\nu}' &= \begin{bmatrix} \nu'_1 & \nu'_2 & \dots & \nu'_m \\ \nu'_1 & \nu'_2 & \dots & \nu'_m \\ \vdots & \vdots & & \vdots \\ \nu'_1 & \nu'_2 & \dots & \nu'_m \end{bmatrix} \\
\text{linewidth} = \Delta\boldsymbol{\nu} &= \begin{bmatrix} \Delta\nu_1 & \Delta\nu_2 & \dots & \Delta\nu_m \\ \Delta\nu_1 & \Delta\nu_2 & \dots & \Delta\nu_m \\ \vdots & \vdots & & \vdots \\ \Delta\nu_1 & \Delta\nu_2 & \dots & \Delta\nu_m \end{bmatrix} \\
\text{beamfilling} = \boldsymbol{\eta}_{\mathbf{B}} &= \begin{bmatrix} \eta_{\mathbf{B}}(\nu'_1) & \eta_{\mathbf{B}}(\nu'_2) & \dots & \eta_{\mathbf{B}}(\nu'_m) \\ \eta_{\mathbf{B}}(\nu'_1) & \eta_{\mathbf{B}}(\nu'_2) & \dots & \eta_{\mathbf{B}}(\nu'_m) \\ \vdots & \vdots & & \vdots \\ \eta_{\mathbf{B}}(\nu_1)' & \eta_{\mathbf{B}}(\nu_2)' & \dots & \eta_{\mathbf{B}}(\nu_m)' \end{bmatrix} \\
\text{obsfreq(jlabel)} = \mathbf{x}_{\text{selected}} &= \begin{bmatrix} x_{i_1 - \text{range}/2} & x_{i_2 - \text{range}/2} & \dots & x_{i_m - \text{range}/2} \\ \vdots & \vdots & & \vdots \\ x_{i_1} & x_{i_2} & \dots & x_{i_m} \\ \vdots & \vdots & & \vdots \\ x_{i_1 + \text{range}/2} & x_{i_2 + \text{range}/2} & \dots & x_{i_m + \text{range}/2} \end{bmatrix} \\
\boldsymbol{\tau} = \tau(\mathbf{x}_{\text{selected}}) &= \begin{bmatrix} \tau(x_{i_1 - \text{range}/2}) & \tau(x_{i_2 - \text{range}/2}) & \dots & \tau(x_{i_m - \text{range}/2}) \\ \vdots & \vdots & & \vdots \\ \tau(x_{i_1}) & \tau(x_{i_2}) & \dots & \tau(x_{i_m}) \\ \vdots & \vdots & & \vdots \\ \tau(x_{i_1 + \text{range}/2}) & \tau(x_{i_2 + \text{range}/2}) & \dots & \tau(x_{i_m + \text{range}/2}) \end{bmatrix}
\end{aligned}$$

Line 41–43 converts **svalue** into a sparse matrix, **lines**, whose size is equal to `length(obsfreq) × length(molfreq)`.

Line 44 sums up all columns (i.e. all transitions) in **lines**, and returns a full-size vector, which is the final spectrum we want.

References

- [1] Rad M. L., Zou L., Sanders J. L. and Widicus Weaver S. L., “Global Optimization and Broadband Analysis Software for Interstellar Chemistry (GOBASIC).” *Astron. & Astrophys.*, in revision, **2015**