شبیه سازی:

A Stable and Efficient Adaptive Notch Filter for Direct Frequency Estimation

بهمن ماه ۱۳۹۸

فاطمه نورزاد

۸۱۰۱۹۸۲۷۱

بخش اول: مقدمه

در بسیاری از کاربرد ها معمولا با مشکل تخمین فرکانس های یک سیگنال به فرم سینوسی به همراه نویز اضافه شونده روبه رو هستیم. برای برطرف کردن آن ۲ روش پردازش off-line processing و on-line processing پیشنهاد میشود.

روش اول به تخمین بر اساس طیف سیگنال (به طور مثال با استفاده از تبدیل فوریه و یا MUSIC) میپردازد. در این روش فرکانس ها براساس مشخص کردن محل دقیق بیشترین و یا کمترین مقدار طیف محاسبه شده بدست می آیند. مشکل روش های off-line processing محاسبات سنگین و هزینه بر آن هاست.

روش دوم بر اساس تکنیک های adaptive notch filtering است. یکی از مدل های محبوبی که بر این اساس کار میکند و ویژگی های مثبتی نظیر پایداری بهتر، همگرایی سریعتر و محاسبات بهینه تری میباشد، constrained notch filter است. این روش ویژگی های مثبتی نظیر پایداری بهتر، همگرایی سریعتر و محاسبات بهینه تری دارد.

در روش کلاسیک پارامتر های این فیلتر بر اساس چند جمله هایی از تابع تبدیل آن ساخته میشوند. ضرایب آن تابعی از فرکانس های شکافی است که فیلتر در آن ها تقریبا بهره ای ندارد. فرکانس های هدف هم بر اساس تخمینی که از تابع تبدیل بدست آمده محاسبه میشوند. به این روش ها indirect frequency method گفته میشود. اما مشکلی که این تکنیک ها دارند این است که به ازای هر وفق دادن، پایداری مدل باید مورد بررسی قرار گیرد. این بررسی ها خود باعث افزایش حجم محاسبات خواهد شد.

یکی از روش ها برای برطرف کردن این مشکلات استفاده از ساختار های lattice است که در adaptive IIR filtering problem از آن استفاده شده و به این ترتیب دیگر نیازی به بررسی پایداری مدل نیست.

اما یکی دیگر از روش ها این است که به صورت مستقیم فرکانس ها را در الگوریتم بدست آوریم. به این صورت که لازم است مدل فیلتر از ابتدا براساس آن ها ساخته شود. در مقاله ای که توسط Chen *et al* منتشر شده، نشان داده شده که فیلتر هایی که به صورت مسقیم بر اساس فرکانس ها ساخته شده اند ویژگی های مثبت زیادی (مانند ایده آل بودن فیلتر به علت محل قرارگیری صفر ها، سرعت بالاتر در همگرایی و همچنین عدم نیاز به بررسی پایداری در هر مرحله) دارند. اما همان طور که جلوتر بررسی میشود این الگوریتم هم ضعف های مختص به خود را دارد. یکی ار مهم ترین آن ها این است که در مرحله یادگیری همواره لازم است این فرکانس ها به ضرایب تابع تبدیل، تبدیل شوند. این تبدیل میتواند باعث مشکلات زیادی شود. در واقع این کار علاوه بر این که نیاز به محاسبات بیشتری دارد، اکثر خصوصیت های مثبتی که برای مدل فرکانسی بیان کردیم را در حالتی که دقت محاسبات محدود است (حالتی که در تمام زمان هایی که محاسبات ما به صورت real-time است را شامل میشود مثلا هنگام استفاده از DSP ها.) از دست میدهد.

همان طور که در قسمت های بعدی نشان داده خواهد شد، در این حالات امکان دارد فیلتر دیگر خصوصیت ایده آل بودن و حتی پایداری را از دست بدهد.

در این مقاله برای برطرف کردن تمام مشکلات بیان شده، یک الگوریتم فیلتر وفقی جدید مورد بررسی قرار میگیرد که آن هم بر اساس محاسبه و تخمین مستقیم فرکانس است. ایده ی آن این است که دیگر ضرایب تابع تبدیل محاسبه نشوند از خود فرکانس ها برای تمامی محاسبات استفاده شود. این اتفاق را میتوان با تبدیل کردن یک فیلتر مرتبه ۲ n به n فیلتر مرتبه ۲ تحقق بخشید. این الگوریتم در مقابل الگوریتم Chen *et al* (از این به بعد با نام CYL که ابتدا اسم نویسندگان آن است بیان خواهد شد) برتری های زیر را دارد:

- ساده تر است بنابراین پیاده سازی آن بهینه تر خواهد بود.
- الگوریتمی مقاوم تر است بنابراین تمام ویژگی های الگوریتم قبلی را خواهد داشت حتی اگر دقت محدود باشد.

در ادامه ابتدا الگوریتم CYL معرفی شده و مشکلات مورد بررسی دقیقتری قرار میگیرد. به این ترتیب علت رسیدن به الگوریتم جدید روشن تر میشود. سپس الگوریتم جدید معرفی شده و رفتار و همگرایی و پایداری آن مورد بررسی قرار میگیرد. پس از آن برای نشان دادن برتری ها با استفاده از متلب نتایج ۲ مثال مورد بررسی قرار میگیرد.

بخش دوم: الگوریتم CYL و ایده های آن:

فرض میکنیم y(t) یک سیگنال قابل اندازه گیری باشد که از n سیگنال سینوسی s(t) و یک نویز اضافه شونده e(t) تشکیل شده باشد.

$$y(t) = \sum_{k=1}^{n} A_k \cos(\theta_k^o t - \phi_k) + e(t) = s(t) + e(t)$$

در این رابطه s(t) و e(t) مستقل اند. هدف ما محاسبه فرکانس های θ_k^o است به طوری که فقط y(t) در دسترس باشد.

براساس روشی که Rao and Kung بیان کرده اند یک فیلتر شکافی را میتوان به صورت زیر نوشت:

$$H(z^{-1}) = \frac{A(z^{-1})}{A(\alpha z^{-1})} \quad . \; < \alpha < 1$$

$$A(z^{-1}) = 1 + a_1 z^{-1} + \cdots + a_n z^{-n} + \cdots + a_{2n} z^{-2n}$$

$$= \prod_{k=1}^{n}(1 - 2z^{-1}\cos\theta_k + z^{-2}) = \prod_{k=1}^{n} A_o(\theta_k . z^{-1})$$

$$. A_o(\theta_k . z^{-1}) = 1 - 2z^{-1}\cos\theta_k + z^{-2}$$

- **مسئله اول:**

به راحتی میتوان دید که صفر های تابع $H(z^{-1})$ روی دایره واحد هستند. به این معنی که در فرکانس های θ_k هیچ توانی منتقل نمیشود. رفتار فیلتر به اندازه α هم وابسته است. این مقدار null bandwidth را کنترل میکند. در واقع پهنای باند به صورت تقریبی برابر $\pi(1 - \alpha)$ است. واضح است که اگر α به ۱ نزدیک باشد مثلا ۰,۹۹۵ فیلتر بدست آمده به صورت فیلتر ایده آل رفتار میکند.

اگر ضرایب تابع تبدیل را در برداری بریزیم داریم :

$$\hat{\boldsymbol{a}} \triangleq (a_1\ a_2\ \dots\ a_k\ \dots\ a_n)^T$$

در الگوریتم Nehorai این n پارامتر بالا آپدیت میشوند. بر اساس روابطی که در بالا بیان شده تابع تبدیل را بر اساس فرکانس هم میتوان نوشت. بنابراین اگر این فرکاس ها را نیز در برداری بریزیم داریم:

$$\hat{\boldsymbol{\theta}} \triangleq (\theta_1\ \theta_2\ \dots\ \theta_k\ \dots\ \theta_n)^T$$

همچنین با استفاده از روابط بالا دیده میشود که تابع تبدیل ۲ n صفر روی دایره واحد و ۲ n قطب هم روی دایره ای به شعاع α دارد.

حال اگر از الگوریتم Nehorai استفاده کنیم با این تفاوت که فرکانس ها مستقیم در آن محاسبه شوند، به الگوریتم CYL میرسیم. (شبه کد آن به طور کامل در مقاله موجود است.)

در این الگوریتم از Gaussian-Newton که برای بدست آوردن خطای تخمین به صورت بازگشتی است، وام گرفته شده. این الگوریتم ویژگی های خوبی نظیر :

صفر های فیلتر به صورت اتوماتیک روی دایره واحد قرار میگیرند. به این ترتیب میتوان مطمین بود که در فرکانس های شکافی توان انتقالی صفر است. این ویژگی لازمه ایده آل بودن فیلتر است که به خوبی در اینجا برآورده شده است.

همگرایی در این روش بسیار سریع رخ میدهد. به این علت که تعداد مینیم های کلی برای $\hat{\boldsymbol{\theta}}$ زیاد است.

پایداری نیز حتما وجود دارد بنابراین لازم نیست بعد از هر iteration آن را بررسی کرد.

حال باید به این موضوع توجه شود که علاوه بر مسائل بیان شده بالا که نکات مثبت آن هستند، این الگوریتم نکات منفی ای دارد که باعث کم رنگ شدن یا از بین رفتن این نکات مثبت میشود.

همان طور که در شبه کد دیده میشود در این الگوریتم لازم است فرکانس های بدست آمده که در بردار $\hat{\boldsymbol{\theta}}$ قرارر دارند به ضرایب تابع تبدیل موجود در $\hat{a}$ تبدیل شوند. این مورد خود باعث میشود که محاسبات real-time و همچنین

محاسبات با استفاده از پردازنده DSP که حافظه محدودی دارند دچار مشکل شود. درواقع این مراحل علاوه به این که حجم محاسبات را افزایش میدهند باعث میشوند که پایداری تابع تبدیل هنگام استفاده آن ها در FWL دچار مشکل شود. به طور مثال فرض میکنیم بردار

$$\hat{\theta} \triangleq (۰.۰۵\ ۰.۱\ ۰.۵)^T$$

به صورت بالا باشد. به این ترتیب با انجام محاسبات زیر عناصر بردار $\hat{a}$ بدست می آیند.

$$a_۱ = -۲ * (cos\theta_۱ + cos\theta_۲ + cos\theta_۳) = -۵.۷۴۲۷$$

$$a_۲ = ۳ + ۴ * cos\theta_۳ * (cos\theta_۱ + cos\theta_۲) + ۴ * cos\theta_۱ * cos\theta_۲$$
$$= ۱۳.۹۷۳۸$$

$$a_۳ = -۴ * (cos\theta_۱ + cos\theta_۲ + cos\theta_۳ + ۲\, cos\theta_۱ cos\theta_۲ cos\theta_۳)$$
$$= ۱۸.۴۶۲۲$$

حال اگر این فیلتر با استفاده از DSP با ظرفیت ۱۶ بیت ساخته شود، تابع تبدیل ساخته شده با این ضرایب نه تنها پایدار نیست بلکه باعث میشود صفر ها هم دیگر روی دایره واحد نباشند. به بیان بهتر دیگر خواص خوب این الگوریتم از بین میرود.

این مشاهدات ما را تشویق میکنند که به دنبال الگوریتم جدید خود باشیم.

بخش سوم: الگوریتم جدید

در این روش یک الگوریتم جدید مورد بررسی قرار میگیرد که فرکانس ها به طور مستقیم محاسبه شده و دیگر نیازی به محاسبه $\hat{\boldsymbol{a}}$ نیست. ایده اصلی هم این است که به گونه ای این فیلتر ها را سری کنیم که دیگر نیازی به ضرایب $\hat{\boldsymbol{a}}$ نباشد.

فرض میکنیم که تابع تبدیل به صورت زیر تعریف شود:

$$H(z^{-۱}) = \prod_{k=۱}^{n} H_o(\theta_k . z^{-۱})$$

به طوری که :

$$H_o(\theta_k . z^{-۱}) = \frac{۱ - ۲\,\beta z^{-۱} \cos\theta_k + \beta^۲ z^{-۲}}{۱ - ۲\,\alpha z^{-۱} \cos\theta_k + \alpha^۲ z^{-۲}}$$

که α مطابق قسمت قبل انتخاب شده و $\alpha > \beta < ۱$ است. بنابراین β = ۰,۹۹۹ را قرار میدهیم. در این صورت صفر های این فیلتر نسبت به آنچه در قسمت قبل دیدیم اندکی متفاوت خواهد بود. اما با توجه به این که β بسیار نزدیک به ۱ استف تقریبا همان رفتار را شاهد خواهیم بود.

هنگامی که y(t) از فیلتر عبور کند، خروجی آن به صورت: $\hat{e}(t) = H(z^{-۱})y(t)$ است.

$$x_i(t) \triangleq \prod_{k=۱}^{i} H_o(\theta_k \,.\, z^{-۱})y(t)$$

ما معادله های بازگشتی را داریم :

$$x_i(t) = H_o(\theta_i \,.\, z^{-۱})x_{i-۱}(t)$$

که منجر به معادلات فضا-حالت میشود:

$$x_i = (۲(\alpha - \beta)cos\theta_i \quad \beta^۲ - \alpha^۲)Z_i(t) + x_{i-۱}(t)$$

$$Z_i(t+۱) = \begin{pmatrix} ۲\,\alpha cos\theta_i & -\alpha^۲ \\ ۱ & ۰ \end{pmatrix} Z_i(t) + \begin{pmatrix} ۱ \\ ۰ \end{pmatrix} x_{i-۱}(t)$$

$$x_o(t) = y(t)\,.\,\hat{e}(t) = x_n(t)$$

در نتیجه خطای تخمین به صورت بازگشتی قابل محاسبه است .

- **مسئله دوم:**

بر خلاف الگوریتم CYL برای محاسبه ی خطای تخمین از بردار فرکانس به طور مستقیم استفاده شده و ضرایب تابع تبدیل فیلتر مورد نیاز نیست در نتیجه تبدیل فرکانس به ضرایب تابع تبدیل نیاز نیست .بنابراین Z(t) همواره پایدار بوده و صفر های آن اتوماتیک بر روی دایره ای ب شعاع β در هر iteration ، بدون توجه به اینکه پارامتر های ساختن فیلتر چه باشد ، قرار میگیرد . این موضوع باعث میشود حتی وقتی دقت بینهایت نیست ویژگی های مثبت الگوریتم حفظ شود .

تابع هزینه به صورت زیر تعریف میشود:

$$V(\widehat{\theta}\,.\,t) \triangleq \frac{۱}{۲\,t}\sum_{j=۱}^{t} \hat{e}^۲(j)$$

با مینیمم کردن تابع هزینه بر اساس $\widehat{\theta}$ می توان تخمینی از فرکانس های توابع سینوسی وارد شده بدست اورد، این روش بر اساسRPE برای اپدیت کردن فرکانس ها ست.

برای مقایسه ی پارامتر های $\theta(t)$ و P(t-۱) ، به $\hat{e}(t)$ و $\psi_\theta(t-۱)$ نیاز است .

$\hat{e}(t)$ به کمک معادله ی فضا-زمان که در بالا به آن اشاره شد ، قابل محاسبه است چون در زمان t $\{Z_i(t).x_i(t)\}$در اختیار ما قرار دارند .

حال داریم :

$$\psi_\theta(t-1) \triangleq -[\frac{\partial \hat{e}(t)}{\partial \hat{\theta}}]$$

$$\hat{e}(t) \triangleq \prod_{k=1}^{n} H_o(\theta_k . z^{-1}) y(t)$$

و در ادامه داریم :

$$\frac{\partial \hat{e}(t)}{\partial \theta_i} = \frac{\partial H_o(\theta_k . z^{-1})}{\partial \theta_i} \prod_{k \neq i} H_o(\theta_k . z^{-1}) y(t)$$

$$= \frac{\partial H_o(\theta_i . z^{-1})}{\partial \theta_i} H_o{}^{-1}(\theta_k . z^{-1}) \hat{e}(t)$$

و در نظر داریم که :

$$\frac{\partial A_o(\theta_i . \gamma z^{-1})}{\partial \theta_i} = 2\gamma z^{-1} sin\theta_i \quad .\gamma = \alpha . \beta$$

بنابراین خواهیم داشت:

$$\frac{\partial \hat{e}(t)}{\partial \theta_i} = 2\left[\frac{\beta z^{-1}}{A_o(\theta_i . \beta z^{-1})}\hat{e}(t) - \frac{\alpha z^{-1}}{A_o(\theta_i . \alpha z^{-1})}\hat{e}(t)\right] sin\theta_i$$

$$= 2\left[e_{F_i}(\beta . t) - e_{F_i}(\alpha . t)\right] sin\theta_i$$

به طوری که داشته باشیم:

$$e_{F_i}(\gamma . t) = [\gamma z^{-1} / A_o(\theta_i . \gamma z^{-1})]\hat{e}(t)$$

در این صورت روابط بالا به صورت فضا-حالت زیر قابل خلاصه شدن است:

$$e_{F_i}(\gamma . t) = (\gamma \quad o) W_i(\gamma . 1)$$

$$W_i(\gamma . t+1) = \begin{pmatrix} 2\alpha cos\theta_i & -\alpha^2 \\ 1 & o \end{pmatrix} W_i(\gamma . t) + \begin{pmatrix} 1 \\ o \end{pmatrix} \hat{e}(t)$$

به طوری که : $\gamma = \alpha . \beta$

(شبه کد این الگوریتم در مقاله موجود است .)

- **مسئله ی سوم**

- انتخاب α مسئله ی بسیار مهمی است ، چراکه پهنای باند کاملا به مقدار ان وابسته است. هنگامی که به یک نزدیک است میتوان به فیلتر ایده ال رسید، اما اگر هیچ اطلاعات اولیه ای در رابطه با مقدار فرکانس ها در دست نباشد، فرکانس های بدست آمده ممکن است با فرکانس های اصلی تطابق نداشته باشد . به طور مثال اگر پهنای باند خیلی باریک باشد باعث میشود الگوریتم حضور موج سینوسی را احساس نکند. به همین علت است که از α متغییر با زمان استفاده میکنیم که از یک مقدار کوچک شروع شده و در طول الگوریتم زیاد میشود . به این ترتیب که یک مقدار اولیه برابر ۰,۹۹ برای آن در نظر گرفته و مقدار نهایی آن هم ۰,۹۹۵ در نظر گرفته میشود. در لحظه ی اول هم مقدار آن را ۰,۱ در نظر میگیریم تا به این ترتیب به پهنای باندی متغییر با زمان برسیم.
- λ برای تخمین پارامتر ها با استفاده از جدید ترین داده استفاده میشود. بنابراین تغییرات آن را مانند α در نظر میگیریم. این رابطه ی بازگشتی کمک میکند که سیگنال های تصادفی مورد پردازش قرار بگیرند.
- در این الگوریتم خطای آینده استفاده شده تا متغییر را به روز رسانی کند. استفاد از آن به این علت است که سرعت همگرایی را بهبود ببخشد، چراکه این خطا ها تخمین بهتری از خطاهای اولیه هستند.

یکی از تفاوت های بین الگوریتم ما و CYL این است که از فضا-حالت استفاده شده است. این روش کمک میکند ک اخرین تخمین ها برای تخمین های بعدی استفاده شود. درواقع خطای تخمین تنها به زمان حال بستگی دارد در حالی که در CYL خطای تخمین به زمان های گذشته نیز وابسته است.

بخش چهارم: انالیز رفتار ها و مقایسه

- **انالیز همگرایی:**

به شکل واضحی الگوریتم بر اساس خطای بازگشتی است . امالیز همگرایی ان نیز باید بر همین اساس باشد. بر اساس برخی فرض ها ، نشان داره میشود که این الگوریتم به مینیمم محلی تابع هزینه یا به مرز مدل همگرا میشود . این فرض ها معمولا ضعیف هستند و برای همه ی فیلتر های این نوع برقرار میباشند . بنابراین انالیز های همگرایی برای این مدل ما برقرار است.

برای امواج سینوسی که نویز اضافه شونده دارند تنها نقطه ی همگرایی این است که به پارامتر های واقعی همگرا شوند به همین علت ما به فرکانس های دقیق همگرا خواهیم شد .

سرعت همگرایی مسئله ی بسیار مهمی است . Nehorai نشان داد که الگوریتمی که براساس ضریب تایع تبدیل است سریعتر از الگوریتم های قدیمی تر همگرا میشود.

بنابراین واضح است که الگوریتم ما و CYL که بر اساس فرکانس هستند با سرعت بیشتری همگرا خواهند شد چراکه همانطور ک بیان شد تابع هزینه ک بر حسب فرکانس است مینیمم های کلی بیشتری دارد که بردار فرکانس ب یکی از ان ها با سرعت همگرا خواهد شد.

ناهمبستگی بین پارامتر ها موردی است که بسیار مورد توجه ماست و این موضوع با استفاده از برسی ماتریس همبستگی مشتق خطای تخمین بدست می اید . برای رسیدن به همگرایی با سرعت بالا نیاز است که این ماتریس قطری بوده و ترجیحا عناصر روی قطر ان به یک میل کند با توجه به تعاریف فرکانس و ضرایب تابع تبدیل این اتفاق برای ماتریس همبستگی بر اساس فرکانس بهتر رخ داده و در نتیجه همگرایی سرعت بیشتری دارد .

- **انالیز پایداری**

همانطور ک گفته شد میخواهیم که مدل در هر iteration پایدار باشد . این یک مسئله ی جی است چرا که ناپایداری میتواند باعث عدم کارکرد الگوریتم شو ، بنابراین بررسی الگوریتم در هر iteration لازم است .

این بدان معنی است که در هر بار لاز است پایداری چک شود که همانطور که گفه شد حجم محاسبات را بالا برده و زمان بیشتری طول میکشد . در الگوریتم CYL و الگوریتم بیان شده ی ما نیازی به بررسی پایداری نیست چراکه مدل به طور اتوماتیک پایدار خواهد بود. این ویژگی برای الگوریتم CYL در صورتی برقرار است که دقت بی نهایت باشد اما برای الگوریتم ما همواره برقرار خواهد بود .(در قسمت های قبلی به طور کامل توضیح داده شده .)

- **Cramer – Rao Bound**

این باند در واقع یک حد پایین برای ماتریس کوواریانس است، که بر اساس خطای تخمین بدست می آید:

$$cov(\hat{\theta}) = E\left[(\hat{\theta} - \hat{\theta}_o)(\hat{\theta} - \hat{\theta}_o)^T\right] \geq J_\theta^{-1}$$

به J_θ^{-1} ماتریس داده Fisher میگوبند.

y(t) یک فرایند ARMA است که برای آن این ماتریس به صورت زیر خواهد بود:

$$J_\theta = \frac{N}{\sigma_e^2} E\left[\psi_\theta(t)\psi_\theta^T(t)\right] . \sigma_e^2 = E[e^2(t)]$$

انتظار داریم که الگوریتم ما به این حد پایین برسد.

اگر داشته باشیم:

$$R = \{r_{kl}\} = \frac{1}{\sigma_e^2} E\left[\psi_\theta(t)\psi_\theta^T(t)\right]$$

به طوری که:

$$r_{ij} = E[\left(\frac{\partial \hat{e}(t)}{\partial \theta_i}\right)\left(\frac{\partial \hat{e}(t)}{\partial \theta_j}\right)]/\sigma_e^2$$

در این صورت داریم:

$$r_{kl} = \frac{1}{j2\pi}\oint \frac{\partial H_o(\theta_k . z^{-1})}{\partial \theta_k} H_o^{-1}(\theta_k . z^{-1}) \cdot \frac{\partial H_o(\theta_l . z)}{\partial \theta_l} H_o^{-1}(\theta_l . z) z^{-1}\, dz$$

$$= \frac{1}{j2\pi}\oint T(\theta_k . z^{-1}) T(\theta_l . z) z^{-1} (4\, sin\theta_k sin\theta_l) dz$$

به طور که:

$$T(\theta_k . z^{-1}) = \frac{\partial H_o(\theta_k . z^{-1})}{\partial \theta_k} H_o^{-1}(\theta_k . z^{-1}) = \frac{(\beta - \alpha)(1 - \alpha\beta z^{-1}) z^{-1}}{A_o(\theta_k . \alpha z^{-1}) A_o(\theta_k . \beta z^{-1})}$$

انتگرال بالا روی دابره واحد که در خلاف جهت عقربه های ساعت است. این انتگرال را باید بر اساس قضایای مانده بدست آورد. باتوجه به تبدیلی که بین $\hat{\theta}$ و $\hat{a}$ برقرار است، واضح است که الگوریتم ما از روش های قبلی ساده تر خواهد بود.

- **بهینگی محاسبات**

همانطور که گفته شد در روش های کلاسیک در هر iteration باید پایداری مدل بررسی شود . برای بررسی پایداری باید قطب های مدل را بدست آورد. این باعث افزایش حجم محاسبات میشود.

همانطور که گفته شد CYL و الگوریتم ما نیازی به این بررسی ندارد. به همین علت این الگوریتم ها بهینه تر هستند. در الگوریتم CYL تبدیل فرکانس به ضرایب تابع تبدیل در مراحل میانی الگوریتم نیاز است که این نیز خود باعث افزایش حجم محاسبات نسبت به الگوریتم ما میشود.

برای نشان دادن تمامی موراد بالا مثال های زیر شبیه سازی میشود.

بخش پنجم: مثال های عددی

در تمامی این مثال ها فرض میشود که ۵۰۰ نمونه از y تولید شده است. این تولید نمونه ها ۳۰ بار تکرار شده و میانگین این تکرار ها رسم شده است . همچنین e(t) نویز سفید با میانگین صفر و واریانس ۱ است.

- **مثال اول :**

$$y(t) = ۲\sin(۰.۵\,t) + ۲\sin(t) + ۲\sin(۲\,t) + e(t)$$

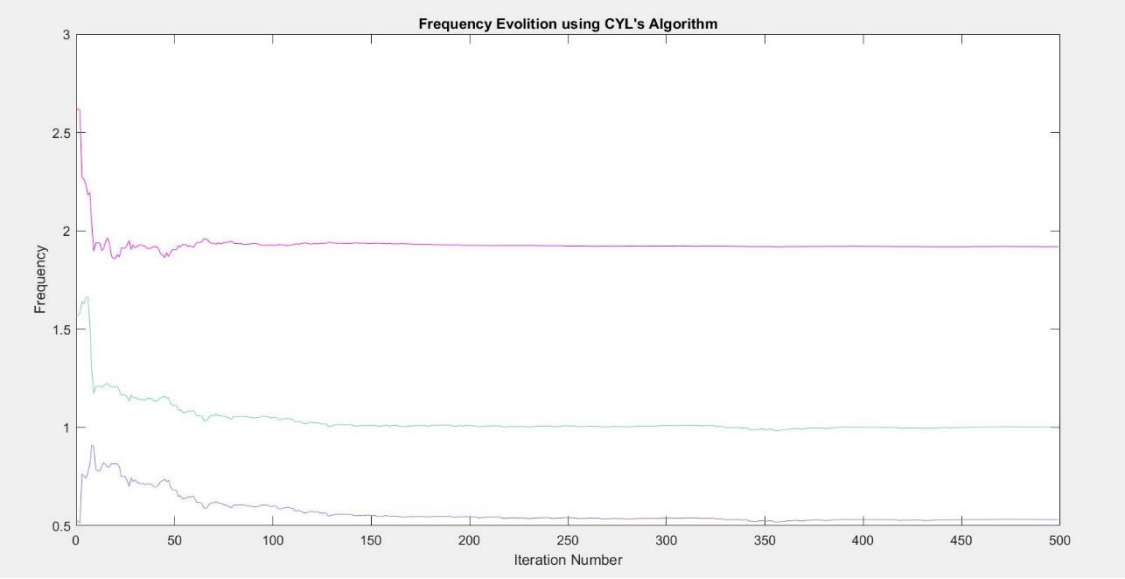
برای این که بررسی عادلانه ای بین ۳ الگوریتم داشته باشیم، مقادیر اولیه زیر را فرض میکنیم:

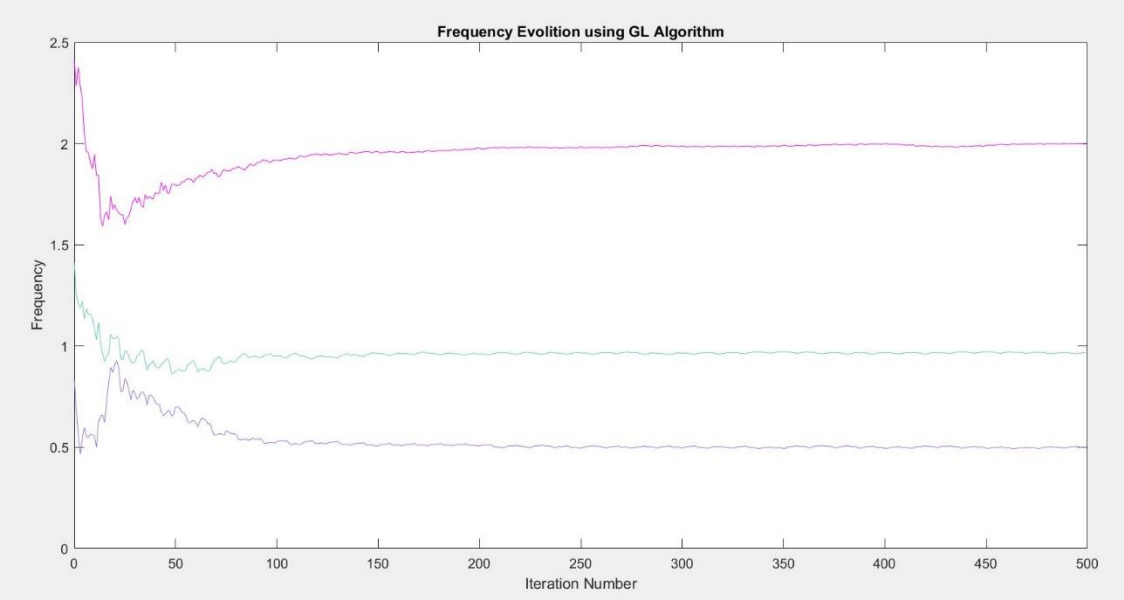
$$\widehat{\boldsymbol{a}}(o) \triangleq (o\ o\ o)^T$$

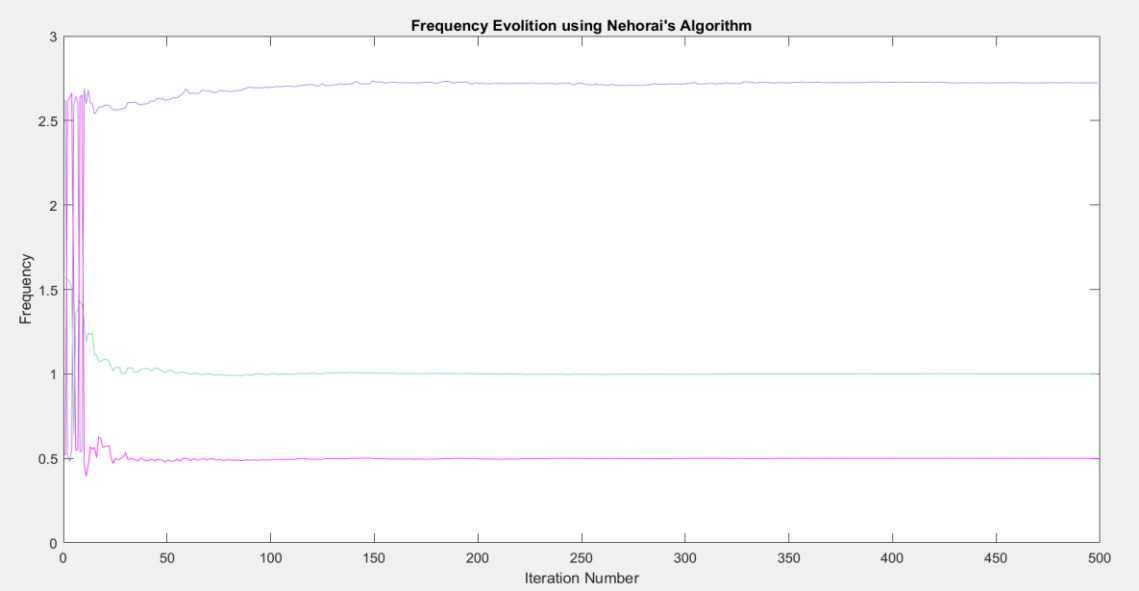
$$\widehat{\boldsymbol{\theta}}(o) \triangleq (۲.۶۱۸\ ۱.۵۷۰۸\ ۰.۵۲۳۶)^T$$

$$P(o) = ۰.۱\ I$$

نتایج شبیه سازی به صورت زیر است:







همان طور که در شبیه سازی های بالا میتوان دید، الگوریتم بیان شده در این مقاله بهتر از دو الگوریتم دیگر همگرا شده است. در راطه با هر سه الگوریتم تعیین پارامتر λ به صورت تجربی و انجام گرفته است. به این علت که در مقاله ها مقادیری برای آن ذکر نشده است.

همان طور که در بخش های قبل توضیح داده شده، حساسیت به پارامتر λ زیاد است که مخصوصا این موضوع در رابطه با الگوریتم های Nehorai و CYL قابل مشاهده است. بر خلاف آنچه در مقاله گفته شده، تنها الگوریتمی که در تمامی ران کردن ها جواب کاملا دقیق و مناسبی میدهد الگوریتم GL است. چراکه به علت دقت محدود متلب $\hat{a}$ به طور دقیق بدست نیامده و همین باعث ناپایداری در دو الگوریتم دیگر میشود.

با این که با روش سعی و خطا سعی شده بهترین مقدار λ یافت شود تا به جواب های منطقی رسید، در برخی موارد به علت دقت محدود متلب دیده میشود که CYL به مقدار درست همگرا نمیشود. در گزارش سعی شده بهترین عکس که به شکل مقاله نزدیک است قرار داده شود که بعد از چند ران میتوان به آن رسید. (در ابتدای مقاله به این موضوعات اشاره شده اما در توضیحات شبیه سازی گفته نشده بود که لازم دانستم بیان کنم.)

بنابراین همین الگوریتم GL بهترین روش است.

- **مثال دوم :**

$$y(t) = ۲\sin(۰.۲۵\,t) + ۲\sin(۰.۷۵\,t) + ۲\sin(t) + e(t)$$

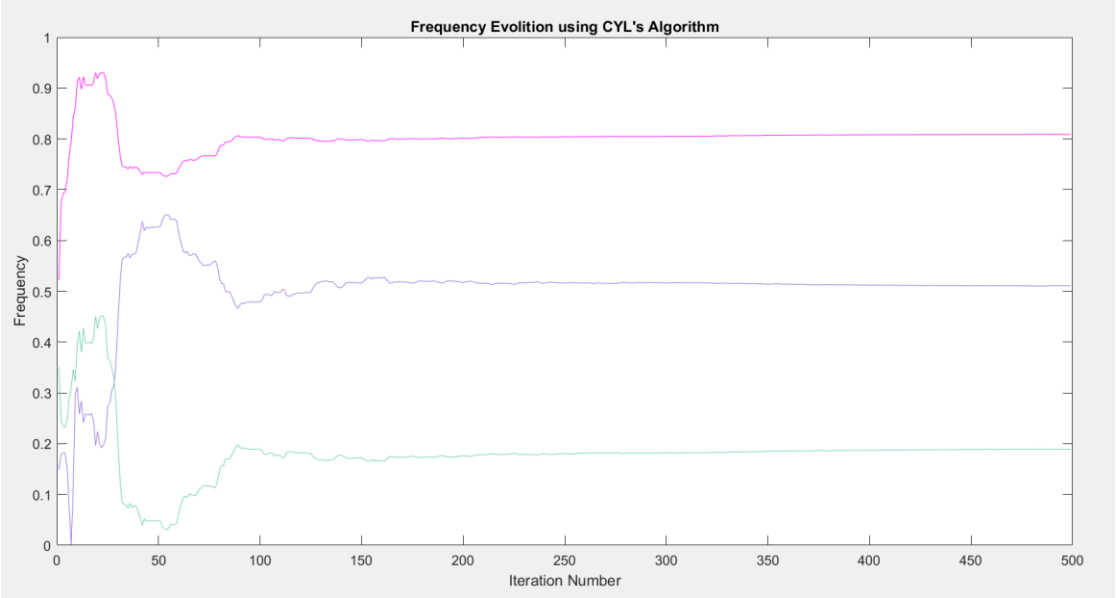
برای این که بررسی عادلانه ای بین ۳ الگوریتم داشته باشیم، مقادیر اولیه زیر را فرض میکنیم:

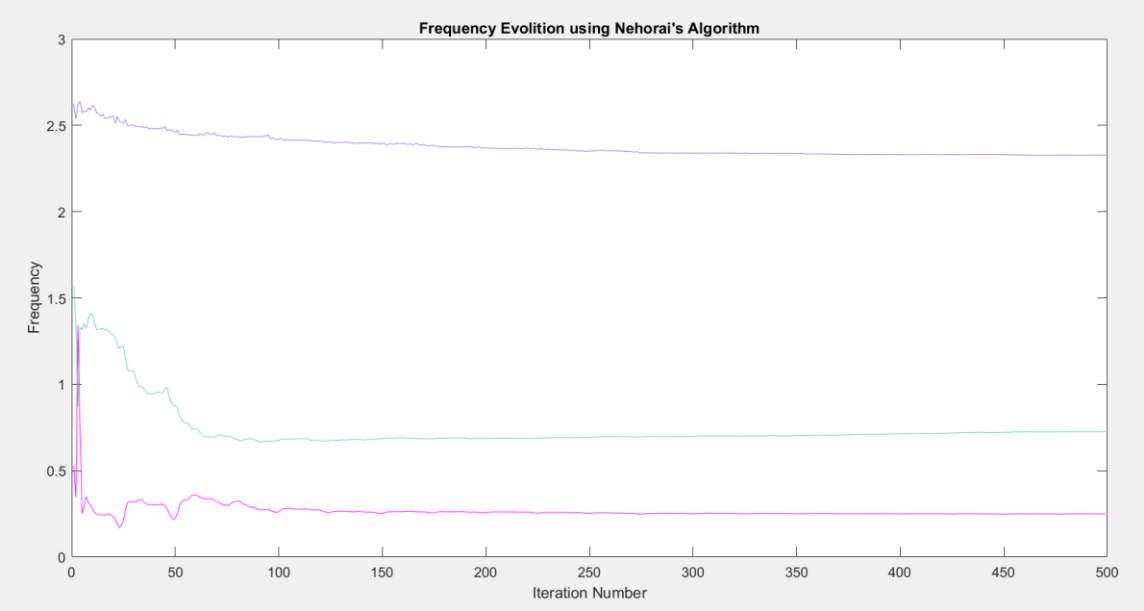
$$\hat{\boldsymbol{a}}(o) \triangleq (o\ o\ o)^T$$

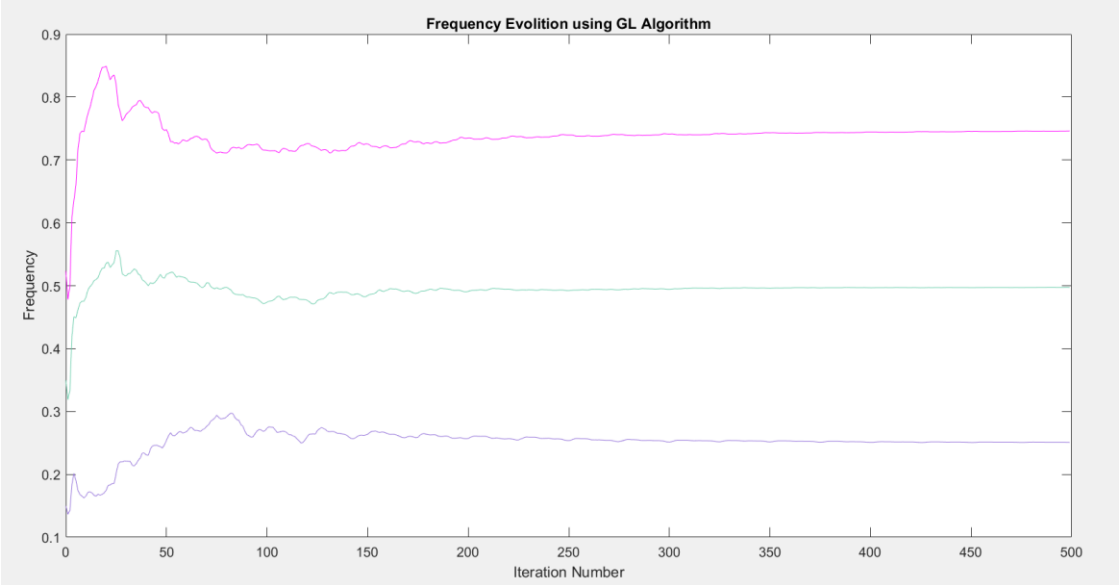
$$\hat{\boldsymbol{\theta}}(o) \triangleq (۰.۱۵\ ۰.۳۵\ ۰.۵۲۳۶)^T$$

$$P(o) = ۰.۱\ I$$

تایج شبیه سازی به صورت زیر است:







با مقایسه اشکال بالا این تفاوت های بیان شده در قسمت های قبل به صورت مشهود تری قابل دیدن است. همان طور که واضح است در مثال دوم الگوریتم Nehorai به مقادیر درست همگرا نشده است و برای CYL هم میتوان دید همگرایی دیرتر از الگوریتم ما رخ داده.

- **توضیحات شبیه سازی ها:**
  - **برای رسیدن از $\hat{\theta}$ به $\hat{a}$:**

در رابطه با این موضوع از روابطی که در قسمت قبل بیان شد استفاده شده است. این تبدیل در یکی از مراحل میانی الگوریتم CYL مورد نیاز بوده است.

- **برای رسیدن از $\hat{a}$ به $\hat{\theta}$:**

استفاده آن در الگوریتم Nehorai است. چراکه در این الگوریتم در هر iteration مقادیر $\hat{a}$ محاسبه میشوند و ما برای رسم در نمودار ها و مقایسه آن ها به مقادیر $\hat{\theta}$ نیازداریم. بنابراین بعد از iteration با استفاده از این ضرایب بدست آمده، تابع تبدیل فیلتر را بدست می آوریم. سپس ریشه های آن را محاسبه مینماییم. کسینوس معکوس قسمت حقیقی ریشه و یا سینوس معکوس قسمت موهومی آن، $\hat{\theta}$ را به ما میدهد.

در برخی از ران ها به علت خطای رخ داده به علت دقت محدود متلب ممکن است ریشه ها دقیق بدست نیایند و در نتیجه مقادیر حقیقی یا موهومی آن ها بزرگتر از ۱ شود. در این حالت ارور در زمان اجرا خواهیم داشت. با دوباره ران کردن این مشکل رفع میشود. (باید دقت شود که این از جمله مشکلات بیان شده برای این الگوریتم در قسمت های قبلی است.)

- **مثال سوم :**

در این قسمت سعی شده گفته شود که میتوان از این الگوریتم برای بررسی صدا هم استفاده کرد. به این ترتیب که فرض میکنیم سیگنال صدا قابل مدل کردن با تعدادی سینوسی است. این توابع سینوسی از یک فرکانس پایه تشکیل شده اند و هریک با دیگر در ضریبی متفاوت اند.

اگر صدا را به این صورت مدل کنیم:

$$y(t) = \sum_{k=۱}^{n} A_k \cos(k\theta_o t - \phi_k) + e(t)$$

در این صورت برای استفاده از الگوریتم لازم است:

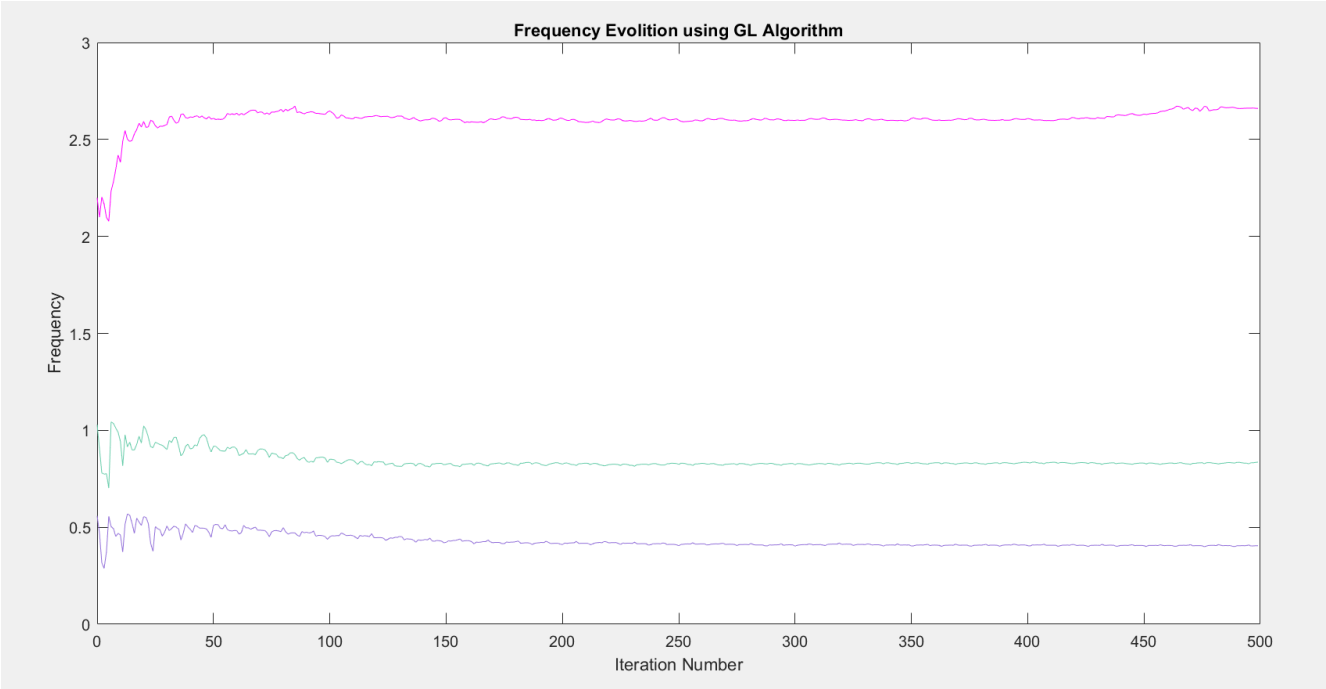
$$\hat{\theta}(t) = (۱\ ۲\ ۳\ \dots k\ \dots n)^T \hat{\theta}_o(t)$$

$$\psi_{\theta_o}(t) = (۱\ ۲\ \dots k\ \dots n)^T \widehat{\psi_\theta}$$

به این ترتیب با این جایگذاری این موارد در الگوریتم، میتوان حتی فرکانس های صدا را هم تشخیص داد. در صورتی که عملا امکان این کار برای سایر الگوریتم ها خیلی سخت میشد.

بخش ششم: بررسی مورد خواسته شده

فرمودید که اگر تعداد سینوسی های داده شده به الگوریتم را تغییر دهیم اما n همان ۳ باقی بماند چه اتفاقی رخ میدهد. برای هر سه الگوریتم، پس از سیمولیت کردن دیده شد که الگوریتم خراب شده و توانی تشخیص فرکانس ها را ندارد. بنابراین برای هر ۳ الگوریتم باید تعداد فرکانس های ورودی را برای تشخیص بدانیم.



به طور مثال شکل بالا نشان دهنده حالت شبیه سازی شده در فایل eg۳GL است. همان طور که دیده میشود فرکانس های دیگر نیز دقت خود را از دست داده اند.