تمرین شماره5

علیرضا حسینی

شماره دانشجویی : ۸۱۰۱۰۱۱۴۲

جداسازی کور منابع

دکتر اخوان

بهار 1402

فهرست مطالب

فهرست اشکال

۱-۱- مقدمه

بازیابی Sparse signals تکنیکی است که در پردازش سیگنال، سنجش فشاری و یادگیری ماشین برای تخمین سیگنال Sparse یا بردار ضریب از تعداد کمی از اندازه‌گیری‌های خطی استفاده می‌شود. هدف بازیابی سیگنال یا بردار ضریب اصلی با استفاده از اندازه‌گیری‌های کمتری است.

روش‌های مختلفی برای بازیابی Sparse signals وجود دارد، از جمله انتخاب زیرمجموعه، تعقیب منطبق (MP)، تعقیب تطبیق متعامد (OMP)، و حداقل مربعات با وزن مجدد (IRLS). این روش ها از نظر پیچیدگی، دقت و کاربرد متفاوت هستند.

انتخاب زیرمجموعه یک رویکرد brute-force است که شامل حل تمام سیستم‌های خطی Sparse signals ممکن برای هر زیر مجموعه ممکن از ستون‌ها در ماتریس Dictionary است. در حالی که راه حل دقیقی را ارائه می دهد، اما از نظر محاسباتی گران و برای دیکشنری های بزرگ غیر عملی است.

Matching Pursuit (MP) یک الگوریتم حریصانه است که با مجموعه ای خالی از اتم های انتخاب شده شروع می شود و به طور مکرر اتم هایی را اضافه می کند که بیشترین همبستگی را با خطای باقی مانده ارائه می دهد. MP را می توان با یک Dictionary ثابت یا تطبیقی پیاده سازی کرد و می تواند دقت خوبی برای سطوح پراکندگی متوسط ارائه دهد.

تعقیب تطبیق متعامد (OMP) گونه‌ای از MP است که متعامد بودن بین اتم‌های انتخاب‌شده و خطای باقی‌مانده را حفظ می‌کند. OMP می تواند دقت بهتری نسبت به MP ارائه دهد و از نظر محاسباتی کارآمدتر است.

حداقل مربعات با وزن مجدد تکراری (IRLS) یک روش تکراری است که با به روز رسانی وزن ها بر اساس برآورد فعلی بردار ضریب پراکندگی، مسئله حداقل مربعات وزنی را به حداقل می رساند. IRLS می تواند نویز غیر گاوسی را کنترل کند و می تواند دقت خوبی برای سطوح پراکندگی بالا ارائه دهد.

به طور کلی، انتخاب روش به مشکل خاص و الزامات مربوط به دقت، پیچیدگی و منابع محاسباتی بستگی دارد. انتخاب زیرمجموعه زمانی قابل استفاده است که سطح پراکندگی کوچک باشد و ماتریس Dictionary خیلی بزرگ نباشد. MP و OMP می توانند دقت خوبی ارائه دهند و از نظر محاسباتی کارآمد هستند، در حالی که IRLS می تواند نویز غیر گاوسی و سطوح پراکندگی بالا را کنترل کند.

Subset Selection -۱-۲

1-2-1- مقدمه

انتخاب زیر مجموعه (Subset Selection) تکنیکی است که در روش های بازیابی sparse سیگنال ها برای تخمین بردار ضریب پراکندگی از بردار مشاهده و ماتریس Dictionary استفاده می شود. هدف روش انتخاب زیرمجموعه انتخاب زیرمجموعه بهینه ستون‌ها از ماتریس Dictionary است که با توجه به محدودیت sparsity ، بردار ضریب sparse را به بهترین شکل نشان می‌دهد.

در انتخاب زیر مجموعه، با انتخاب همه زیرمجموعه‌های ممکن ستون‌ها در ماتریس Dictionary با سطح sparsity ثابت شروع می‌کنیم. برای هر زیر مجموعه، سیستم خطی معادلات مربوطه را حل می کنیم تا تخمینی از بردار ضریب sparsity به دست آوریم. در نهایت، تخمینی را انتخاب می‌کنیم که خطای بین بردار مشاهده و بردار پیش‌بینی‌شده به‌دست‌آمده از زیر مجموعه ستون‌های انتخابی را به حداقل می‌رساند.

روش انتخاب زیرمجموعه به طور گسترده در کاربردهای مختلفی مانند پردازش سیگنال، سنجش فشرده

سازی و پردازش تصویر استفاده شده است. این یک روش محاسباتی گران است زیرا نیاز به حل چندین سیستم خطی برای هر زیر مجموعه ممکن از ماتریس Dictionary دارد.

2-2-1- پیاده سازی

ابتدا باید داده ها لود شود که این کار به کمک کد زیر انجام میشود.

```
%% load Data :

load('hw5.mat')
[M, N] = size(D);
```

شکل (۱-۱) کد متلب لود کردن داده ها

اجرای روش subset selection به شرح زیر است:

- سطح پراکندگی N0 را تنظیم کنید و حداکثر خطا را به بی نهایت مقداردهی کنید.
- با استفاده از تابع ()commbnk برای تولید همه زیرمجموعه های ممکن با اندازه N0، روی همه زیر مجموعه های ممکن N0-sparse s تکرار کنید.
- برای هر زیر مجموعه، بررسی کنید که آیا ستون‌های ماتریس Dictionary مربوط به زیرمجموعه مستقل هستند یا نه. اگر آنها نیستند، به زیر مجموعه بعدی بروید.
- سیستم خطی Dsubset*s_temp=x را برای سیگنال پراکنده s_temp با استفاده از زیر مجموعه انتخاب شده از ماتریس Dictionary حل کنید.
- خطای بین مشاهده و تقریب مشاهده را با استفاده از زیر مجموعه انتخاب شده محاسبه کنید.
- اگر خطای محاسبه شده در مرحله قبل کوچکتر از حداکثر خطا باشد، سیگنال پراکنده s و حداکثر خطا را به روز کنید.

- پس از بررسی همه زیر مجموعه های ممکن، سیگنال پراکنده s با کوچکترین خطا انتخاب می شود.
- عناصر غیر صفر و موقعیت آنها در سیگنال پراکنده و همچنین زمان اجرا را چاپ کنید.

کد ارائه شده در بالا روش انتخاب زیر مجموعه را برای بازیابی پراکنده پیاده سازی می کند. ابتدا سطح پراکندگی را روی N0=3 تنظیم می کند و با استفاده از تابع tic() تایمر را شروع می کند. سپس با استفاده از تابع commbnk() روی تمام زیر مجموعه‌های ممکن با اندازه N0 از ستون‌های ماتریس دیکشنری D تکرار می‌شود. برای هر زیرمجموعه، کد بررسی می کند که آیا ستون های ماتریس Dictionary مربوط به زیرمجموعه به صورت خطی مستقل هستند یا خیر. در صورت وجود، کد سیستم خطی را با استفاده از زیر مجموعه انتخاب شده حل می کند و خطای بین مشاهده و تقریب مشاهده را محاسبه می کند. اگر خطا کمتر از حداکثر خطای محاسبه شده تا کنون باشد، کد سیگنال پراکنده s و حداکثر خطا را به روز می کند. پس از بررسی تمام زیرمجموعه های ممکن، کد عناصر غیر صفر و موقعیت آنها را در سیگنال پراکنده و همچنین زمان اجرا را با استفاده از تابع disp() و تابع toc() چاپ می کند.

```
%% Subset Selection Method , minimize Norm 0
N0 = 3;
tic;
% iterate over all possible N0-sparse subsets of s
max_err = Inf;
for subset = combnk(1:N, N0).' % generate all possible subsets of size N0
    Dsubset = D(:, subset);
    if rank(Dsubset) < N0 % skip linearly dependent subsets
        continue;
    end
    s_temp = Dsubset \ x;
    err_temp = norm(x - Dsubset * s_temp);
    if err_temp < max_err % update if found a better subset
        s = zeros(N, 1);
        s(subset) = s_temp;
        max_err = err_temp;
    end
end
disp(['Subset Selection , minimize norm 0 : s = [' num2str(s(:).') '], error = ' num2str(max_err)])
nz_idx = find(s);
disp(['Non-zero elements: ' num2str(s(nz_idx).')])
disp(['Positions: ' sprintf('%d ', nz_idx)])
toc;
```

شکل (۲-۱) کد متلب subset selecton

خروجی کد به شرح زیر است :

```
Subset Selection , minimize norm 0 : s = [0          0          5          0          0          7
Non-zero elements: 5          7          -3
Positions: 3 6 54
Elapsed time is 0.308389 seconds.
>>
```

شکل (۳-۱) خروجی کد subset selection

که در کد فوق سیگنال s و مقادیر non zero و پوزیشن آن ها و زمان اجرای کد را مشاهده میشود.

لازم به ذکر است با توجه به اینکه در این روش به تعداد تمام حالت های انتخاب N0 از N تکرار میشود نیاز میشود که N0 را بدانیم.

میزان error در این حالت error = 1.9984e-15 میباشد.

۳-۱- استفاده از نرم 2 به جای 0

1-3-1- مقدمه

رایج ترین رویکرد در انتخاب زیرمجموعه، به حداقل رساندن نرم 0 است که تعداد عناصر غیرصفر را در بردار راه حل s شمارش می کند. با این حال، به حداقل رساندن L0-norm یک مشکل NP-hard است و از نظر محاسباتی برای مسائل در مقیاس بزرگ غیرممکن است.

میشود از L2-norm برای ارزیابی خطای هر زیرمجموعه کاندید استفادهکرد ، که ممکن است غیرمعمول به نظر برسد زیرا L2-norm نشانگر خوبی برای پراکندگی نیست. با این حال، این یک رویکرد عملی است که در بسیاری از موارد به خوبی جواب می دهد.(ولی هیچگاه جوابی Sparse نمیدهد) علاوه بر این، ما فقط از هنجار L2 برای ارزیابی زیرمجموعه های کاندید استفاده می کنیم و راه حل نهایی با استفاده از معکوس زیر

مجموعه انتخاب شده از ماتریس دیکشنری D به دست می آید.

$$S_{hat} = Pinv(D) * X$$

2-3-1- پیاده سازی

برای پیاده سازی از کد زیر خیلی ساده به کمک دستور فوق استفاده میشود.

```
%% L2 norm, minimize ||s||_2^2
tic;
s = pinv(D) * x;
err = norm(x - D * s);
disp(['L2 norm, minimize ||s||_2^2: s = [' num2str(s(:).') '], error = ' num2str(err)])
nz_idx = find(s);
disp(['Non-zero elements: ' num2str(s(nz_idx).')])
disp(['Positions: ' sprintf('%d ', nz_idx)])
toc;
% plot s_hat
figure;
stem(s);
title('Recovered sparse signal (L2 norm)');
% check if s is sparse
figure;
stem(abs(s));
title('Magnitude of recovered sparse signal (L2 norm)');
```

شکل (۴-۱) کد متلب مینیمم کردن نرم 2

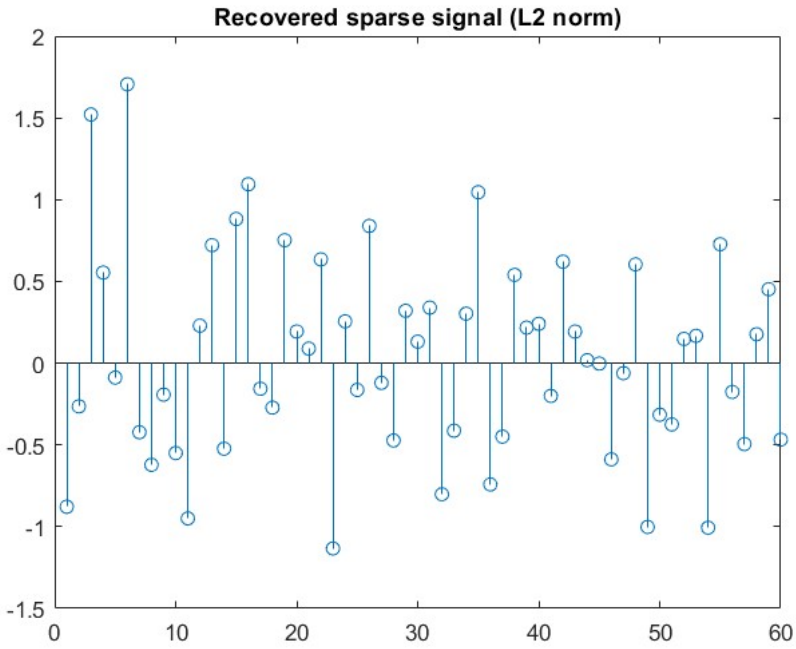
مشاهده میشود سیگنال recover شده اصلا sparse نمیباشد.

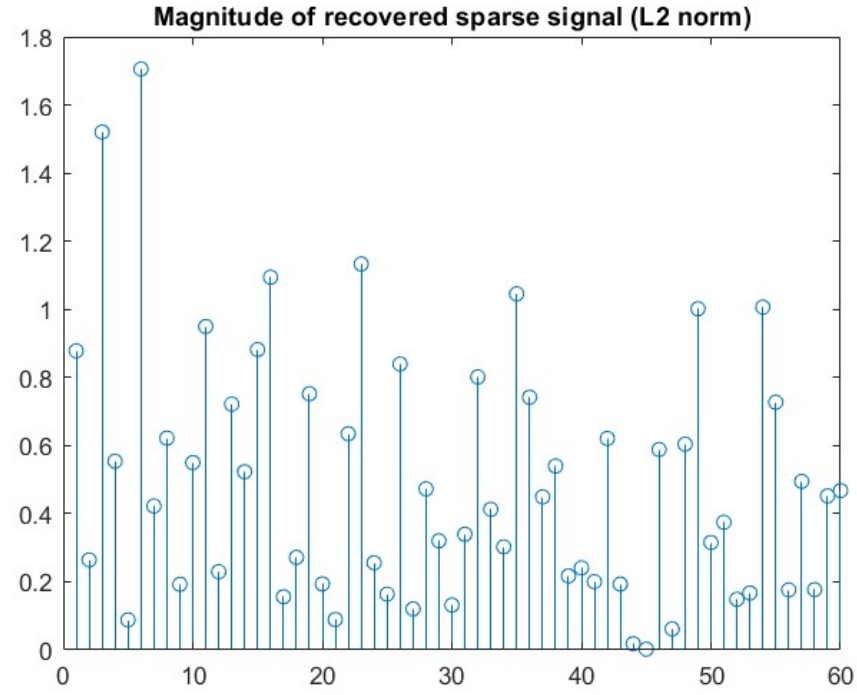
اما Elapsed time is 0.027895 seconds که نشان میدهد این روش نسبت به قبل بسیار سریع تر میباشد.

سیگنال ریکاور شده و اندازه در شکل های زیر آمده است.

لازم به ذکر است در این روش هیچ نیازی به دانستن N0 نمیباشد چراکه اصلا sparse خروجی نمیدهد و sparsity level هم نیازی ندارد.

همچنین مقدار خطا با این روش نیز error = 6.8367e-15 میباشد که بیشتر از حالت قبل است.





شکل (۵-۱) سیگنال و Magnitude سیگنال ریکاور شده با استفاده از مینیمم کردن نرم 2

MP -۱-۴

1-4-1- مقدمه

Matching Pursuit (MP) یک الگوریتم محبوب در پردازش سیگنال و سنجش فشرده برای بازیابی سیگنال پراکنده است. ایده اصلی پشت MP این است که به طور مکرر پشتیبانی سیگنال پراکنده و ضرایب غیر صفر مربوطه را شناسایی کنیم.

الگوریتم با تخمین اولیه سیگنال پراکنده شروع می شود که معمولاً روی صفر تنظیم می شود. در هر تکرار، الگوریتم اتمی از Dictionary را انتخاب می کند که بالاترین حاصلضرب داخلی را با سیگنال باقیمانده دارد و ضریب مربوطه را به تخمین پراکنده اضافه می کند. سپس سیگنال باقیمانده با کم کردن سهم اتم انتخاب شده از سیگنال اصلی به روز می شود.

این فرآیند تا زمانی که یک معیار توقف برآورده شود، تکرار می شود، که می تواند تعداد ثابتی از تکرارها یا آستانه ای در هنجار باقیمانده باشد. خروجی الگوریتم MP تخمینی از سیگنال پراکنده است.

مزیت MP نسبت به سایر الگوریتم های بازیابی پراکنده، مانند انتخاب زیر مجموعه و تعقیب تطبیق متعامد (OMP)، سادگی و هزینه محاسباتی پایین آن است. با این حال، ممکن است در برخی موارد از همگرایی کند باشد یا بهینه نباشد.

1-4-2- پیاده سازی با فرض دانستن N0

در این بخش به ما سطح پراکندگی N0 داده می شود و از الگوریتم Matching Pursuit (MP) برای یافتن تقریب پراکنده x استفاده می کنیم. الگوریتم MP به طور مکرر ستون هایی از D را انتخاب می کند که بیشترین همبستگی را با باقیمانده سیگنال x دارند. در هر تکرار، الگوریتم همبستگی بین سیگنال باقیمانده و هر ستون D را با استفاده از محصول داخلی x1'*D محاسبه می کند.

سپس، ستونی را با حداکثر همبستگی مطلق انتخاب می کند و با تنظیم عنصر مربوطه به عنوان مقدار همبستگی، سیگنال پراکنده s را به روز می کند. سپس سیگنال باقیمانده با کم کردن طرح ریزی x بر روی ستون انتخابی D به روز می شود.

پس از تکرار N0، با استفاده از ستون‌های انتخابی D و ضرایب مربوطه در s، یک تقریب پراکنده برای سیگنال x به دست می‌آوریم. ما شاخص ها و مقادیر غیر صفر تقریب پراکنده و همچنین خطای بین سیگنال اصلی x و سیگنال بازسازی شده D*s را چاپ می کنیم.

```
%% MP ( Known No ) :
N0 = 3;
tic;
x_mp = x;
posMP = zeros(1,N0);
sMP = zeros(N,1);
for i = 1:N0
   ro = x_mp'*D;
   [~,posMP(i)] = max(abs(ro));
   sMP(posMP(i)) = ro(posMP(i));
   x_mp = x_mp - sMP(posMP(i))*D(:,posMP(i));
end
disp('MP for Known N0:');
disp(['Non-zero elements: ' num2str(sMP(posMP).')])
disp(['Positions: ' sprintf('%d ', posMP)])
disp(['Error: ', num2str(norm(x-D*sMP))])
disp(['Run Time: ', num2str(toc), ' seconds'])
```

شکل (۶-۱) کد متلب پیاده سازی MP وقتی N0 را میدانیم.

در این حالت خروجی به صورت زیر میباشد: (شامل S_hat و پوزیشن های غیر صفر و خطا و مدت اجرای الگوریتم)

```
MP for Known N0:
Non-zero elements: 11.1791     -3.75582     -2.05322
Positions: 6 60 9
Error: 2.4506
Run Time: 0.001677 seconds
```

شکل (۷-۱) خروجی MP با دانستن N0

3-4-1- پیاده سازی با فرض ندانستن N0

در این حالت اولا میدانیم که max N0 برابر 10 میباشد و در اینجا همان مراحل قبل را برای N0 برابر 1 تا 10 تکرار میکنیم و یک ترشولد تعریف میکنیم اگر خطا از 1e-1 کمتر شد توقف کند. (این ترشولد بسته به مساله میتواند متفاوت باشد)

```
%% MP ( UnKnown No ) :

N0_max = 10; % maximum expected value of N0
x1 = x;
posMP = zeros(1,N0_max);
sMP = zeros(N,1);
max_err = Inf;
for N0 = 1:N0_max
    ro = x1' * D;
    [~, pos] = max(abs(ro));
    posMP(N0) = pos;
    s_temp = pinv(D(:, posMP(1:N0))) * x;
    err_temp = norm(x - D(:, posMP(1:N0)) * s_temp);
    if err_temp < max_err
        sMP(posMP(1:N0)) = s_temp;
        max_err = err_temp;
    end
    x1 = x - D(:, posMP(1:N0)) * sMP(posMP(1:N0));
    if norm(x1) < 1e-1 % stop iteration if residual is too small
        break;
    end
end
sMP = sMP(posMP(1:N0)); % trim the sMP vector to remove zeros
posMP = posMP(1:N0); % trim the posMP vector to remove zeros
disp('MP with unknown N0:')
disp(['Non-zero elements: ' num2str(sMP.')])
disp(['Positions: ' sprintf('%d ', posMP)])
```

شکل (۸-۱) کد پیاده سازی mp بدون دانستن No

خروجی در این حالت که N0=9 خطای threshold را داشته با تغییر ترشولد میتوان حالت های دیگر را نیز بررسی کرد.

```
MP with unknown N0:
Non-zero elements: 9.4469     -3.7742      2.3093      2.0612     0.92743     0.52218    -0.38381    -0.22105    -0.19252
Positions: 6 60 16 15 56 59 7 17 37
```

شکل (۹-۱) خروجی MP بدون دانستن N0

OMP -۱-۵

1-5-1- مقدمه

(OMP) الگوریتم دیگری برای بازیابی سیگنال پراکنده است. این یک روش تکراری است که با حدس اولیه بردار پراکنده شروع می شود و با افزودن اتم های جدید از Dictionary آن را اصلاح می کند.

الگوریتم OMP دو مرحله اصلی دارد: گام رو به جلو و گام به عقب. در مرحله رو به جلو، OMP با حدس اولیه بردار پراکنده s شروع می کند و محصولات داخلی بین D Dictionary و r=x-D*s باقیمانده را محاسبه می کند. سپس شاخص k اتم با بزرگترین حاصلضرب داخلی مطلق را انتخاب می کند و آن را به مجموعه پشتیبانی T اضافه می کند. در مرحله رو به عقب، OMP ضرایب اتم های مجموعه پشتیبان T را با حل مسئله حداقل مربعات محاسبه می کند و به روز می کند.

الگوریتم OMP این مراحل را تا زمانی ادامه می دهد که خطا باقیمانده به زیر یک آستانه معین بیفتد یا به حداکثر تعداد تکرار برسد. الگوریتم OMP این مزیت را دارد که پیاده سازی سریع و ساده است و می تواند سیگنال های پراکنده را با دقت بالا بازیابی کند.

در عمل، الگوریتم OMP نیاز به آگاهی از سطح پراکندگی N0 سیگنال x از قبل دارد. با این حال، الگوریتم OMP را می توان تغییر داد تا سطح پراکندگی N0 سیگنال x را به طور خودکار برآورد کند. این الگوریتم اصلاح‌شده به الگوریتم آستانه سخت تکراری (IHT) معروف است که از تکنیک آستانه‌گذاری برای حذف کوچک‌ترین ضرایب در هر تکرار استفاده می‌کند تا زمانی که هنجار باقیمانده به زیر آستانه معینی برسد.

2-5-1- پیاده سازی

پارامترهای ورودی الگوریتم شامل ماتریس اندازه گیری D، بردار سیگنال x و تعداد ضرایب غیر صفر در سیگنال اصلی N0 است. الگوریتم با یک بردار تمام صفر به عنوان تخمین سیگنال پراکنده شروع می شود و به طور مکرر عناصر غیر صفر را به تخمین اضافه می کند.

در هر تکرار، الگوریتم همبستگی بین سیگنال باقیمانده و ماتریس اندازه گیری را محاسبه می کند تا اتمی را انتخاب کند که بیشترین همبستگی را با سیگنال باقیمانده دارد. سپس اتم انتخاب شده به مجموعه پشتیبانی جاری، که مجموعه شاخص های ضرایب غیر صفر در برآورد است، اضافه می شود. سپس تخمین با حل یک مسئله حداقل مربعات با استفاده از مجموعه پشتیبانی فعلی به روز می شود. این روند تا زمانی که تعداد ضرایب غیر صفر مورد نظر به دست آید تکرار می شود.

سپس کد شاخص و مقدار عناصر غیر صفر در برآورد و همچنین خطای بازسازی را نمایش می دهد.

```
%% OMP algorithm for known sparsity N0
tic;
N0 = 3
x1 = x;
posOMP = zeros(1,N0);
sOMP = zeros(N,1);
for i=1:N0
   ro = x1'*D;
   [~,posOMP(i)] = max(abs(ro));
   Dsub = D(:,posOMP(1:i));
   sOMP_sub = pinv(Dsub)*x;
   sOMP(posOMP(1:i)) = sOMP_sub(1:i);
   x1 = x - Dsub*sOMP_sub;
end
disp('OMP with known N0 :');
disp(['Non-zero elements: ' num2str(sOMP(find(sOMP~=0))')])
disp(['Error: ' num2str(norm(x-D*sOMP))]);
toc;
```

شکل (۱۰-۱) کد متلب پیاده سازی OMP وقتی NO معلوم است

```
OMP with known N0 :
Non-zero elements: 9.9545      2.5075     -4.1428
Error: 2.1708
Elapsed time is 0.008593 seconds.
```

شکل (۱۱-۱) خروجی OMP با N0 = 3

اما وقتی N0 معلوم نباشد از الگوریتم IHT استفاده میکنیم.

```
%% OMP using IHT algorithm for unknown N0
% Initialize parameters
tol = 1e-4;
max_iter = 1000;
s_omp = zeros(N, 1);
residual = x;

% Main loop
for i = 1:max_iter
    % Compute correlation of residual with dictionary
    corr = abs(D' * residual);

    % Find index with maximum correlation
    [~, index] = max(corr);

    % Add index to support
    support = unique([find(s_omp); index]);

    % Compute least-squares solution for current support
    s_omp(support) = pinv(D(:, support)) * x;

    % Compute new residual
    residual = x - D * s_omp;

    % Check stopping criterion
    if norm(residual) < tol
        break;
    end
end

% Print results
disp('OMP using IHT algorithm for unknown N0:')
```

شکل (۱۲-۱) کد متلب الگوریتم IHT برای OMP

خروجی کد به شرح زیر است :

```
OMP using IHT algorithm for unknown N0:
Number of iterations: 10
Non-zero elements: 10
Non-zero elements: 9.412    -0.32555      2.0965      2.3389    -0.19999     0.13102    -0.14249     0.92231     0.56244     -3.7222
Error: 1.5663e-14
```

شکل (۱۳-۱) خروجی کد الگوریتم IHT

که مشاهده میشود N0 را 10 و مقادیر S ها را مشخص کرده و وقتی N0 =10 باشد بدیهی است که مقدار خطا کمتر میشود.

BP -۱-۶

1-6-1- مقدمه

روش Basis Pursuit (BP) یک رویکرد برنامه ریزی خطی است که برای حل سیستم های خطی نامشخص استفاده می شود. با توجه به یک سیستم معادلات خطی تعریف نشده، که در آن متغیرهای مجهول بیشتری نسبت به معادلات وجود دارد، روش BP پراکنده ترین راه حل را برای سیستم پیدا می کند. این روش شامل به حداقل رساندن نرم 1 است . به عبارت دیگر، روش BP به دنبال یافتن راه حلی است که هم امکان پذیر باشد و هم دارای حداقل پراکندگی باشد. روش BP به طور گسترده در پردازش سیگنال و کاربردهای سنجش فشاری استفاده می شود که هدف آن بازیابی سیگنال از مجموعه کوچکی از اندازه گیری های خطی است.

1-6-2- پیاده سازی

در این متود نیازی به دانستن N0 نمیباشد.

Basis Pursuit (BP) بر اساس مسئله بهینه سازی زیر است:

minimize ||s||_1 subject to y = Ds

مسئله بهینه سازی فوق را می توان با استفاده از حل کننده برنامه خطی به یک مسئله برنامه ریزی خطی تبدیل کرد. تبدیل شامل معرفی یک متغیر کمکی t و محدودیت‌های زیر است:

$$t >= |s\_i| \text{ for } i = 1, 2, ..., N$$

$$y = Ds$$

تابع هدف برنامه خطی به حداقل رساندن t است و حل برنامه خطی سیگنال پراکنده s را به ما می دهد.

در کد MATLAB برای BP با استفاده از برنامه نویسی خطی (linprog) ، ابتدا تابع هدف را تعریف می کنیم که بردارهایی با طول N2 است. سپس ماتریس محدودیت برابری Aeq و بردار سمت راست beq را بر اساس ماتریس D و بردار x تعریف می کنیم. lb کران پایین روی صفر تنظیم شده است.

سپس از تابع linprog MATLAB برای حل برنامه خطی استفاده می کنیم. خروجی linprog یک بردار yhat است که حاوی مقادیر متغیر کمکی t و سیگنال پراکنده s است. سیگنال پراکنده s را از yhat استخراج می کنیم و سپس با استفاده از تابع find موقعیت عناصر غیر صفر را در s پیدا می کنیم.

در نهایت، با استفاده از تابع disp، موقعیت ها و مقادیر عناصر غیر صفر را در s نمایش می دهیم.

```
%% BP using linear programming
tic
f = ones(2 * N, 1); % objective function
Aeq = [D, -D]; % equality constraints
beq = x;
lb = zeros(2 * N, 1); % lower bound for variables
yhat = linprog(f, [], [], Aeq, beq, lb, []); % solve linear program
splus = yhat(1:N); % extract splus and sminus
sminus = yhat(N+1:end);
sBP = splus - sminus; % reconstruct sparse signal
posBP = find(abs(sBP) > 0.01)'; % find non-zero elements
disp('BP:')
disp(['Non-zero elements: ' num2str(sBP(posBP)')]) % display non-zero elements
disp(['Positions: ' num2str(posBP)]) % display positions
disp(['Runtime: ' num2str(toc) ' seconds']) % display runtime
```

شکل (۱۴-۱) کد متلب BP بدون داشتن No

خروجی به صورت زیر است که (N0 را 10 گرفته و خطا نیز بسیار کم است)

```
BP:
Non-zero elements: 5          7          -3
Positions: 3   6  54
Runtime: 0.0096716 seconds
```

شکل (۱۵-۱) خروجی BP بدون داشتن N0

در خروجی به همان جواب بخش 1 (subet selection) رسیدیم ولی با مقدار run time بسیار بسیار کمتر.

Iteratively Reweighted Least Square (IRLS) -۱-۷

1-7-1- مقدمه

(IRLS) یک روش بهینه سازی است که اغلب برای حل مسائل رگرسیون با راه حل های پراکنده استفاده می شود. این به ویژه زمانی موثر است که راه حل به عنوان پراکنده شناخته شود، زیرا می تواند بسیار سریعتر از روش های دیگر که از پراکندگی استفاده نمی کنند، همگرا شود.

IRLS با حل مکرر مسئله حداقل مربعات وزنی کار می کند، جایی که وزن ها با برآورد فعلی راه حل تعیین می شوند. در هر تکرار، راه حل با حل مسئله حداقل مربعات وزنی به روز می شود و وزن ها بر اساس برآورد فعلی راه حل به روز می شوند. این روند تا زمان همگرایی تکرار می شود.

IRLS اغلب همراه با منظم سازی استفاده می شود، که تکنیکی برای کنترل پیچیدگی راه حل با افزودن یک عبارت جریمه به تابع هدف است. رایج ترین شکل منظم سازی، منظم سازی L1 است که مجموع مقادیر مطلق ضرایب را به تابع هدف اضافه می کند. این امر باعث ایجاد پراکندگی در راه حل می شود، زیرا جریمه L1 تمایل دارد بسیاری از ضرایب را به صفر برساند.

IRLS یک ابزار قدرتمند برای رگرسیون پراکنده است و به طور گسترده در بسیاری از زمینه ها از جمله پردازش سیگنال، یادگیری ماشین و آمار استفاده می شود. به ویژه زمانی مفید است که تعداد ویژگی ها در مقایسه با تعداد مشاهدات زیاد باشد، زیرا می توان از آن برای یافتن راه حلی پراکنده که هم دقیق و هم قابل تفسیر باشد استفاده کرد.

2-7-1- پیاده سازی

کد در زیر آمده است. تمامی توضیحات کد کامنت شده است.

```
% initialize weights and set maximum iterations
w = ones(N,1); % initialize weights with ones of size N
ITRmax = 100; % set the maximum iteration

% initialize variables
sIRLS = zeros(N,1); % initialize sIRLS with zeros of size N
sIRLS_prev = zeros(N, 1); % initialize sIRLS_prev with zeros of size N
y = zeros(M,1); % initialize y with zeros of size M
tic

% iterate until maximum iterations or convergence
for itr = 1:ITRmax
W = diag(w); % create a diagonal matrix W with the weight vector w
y = pinv(D*(W^-1))*x; % compute y using the Moore-Penrose pseudoinverse of the product of D and the inverse of W, and the input data x
sIRLS = y./sqrt(w); % update sIRLS with the element-wise division of y by the square root of w

% update weights
for n = 1:N % for each element of the weight vector
    if abs(sIRLS(n)) < eps % if the absolute value of the corresponding sIRLS element is less than the deviation eps
        w(n) = 1e10; % set the weight to a large value
        sIRLS(n) = 0; % set the corresponding sIRLS element to zero
    elseif abs(sIRLS(n)) > 1e6 % if the absolute value of the corresponding sIRLS element is greater than 1e6
        w(n) = 1e-10; % set the weight to a small value
    else % otherwise
        w(n) = 1./abs(sIRLS(n)); % update the weight with the inverse of the absolute value of the corresponding sIRLS element
    end
end

% check for convergence
if norm(sIRLS - sIRLS_prev) < eps % if the L2-norm of the difference between the current and previous sIRLS vectors is less than the deviation eps
    break % exit the loop
else % otherwise
    sIRLS_prev = sIRLS; % update the previous sIRLS vector
end
```

شکل (۱۶-۱) کد متلب IRLS

خروجی به صورت زیر میباشد :

```
Non-zero elements of S:
3 6 54
Values of non-zero elements of S:
2.92 3.66 -2.08
Runtime: 0.0139 seconds
Error: 5.2722
```

شکل (۱۷-۱) خروجی کد IRLS

ران تایم به نسبت زیاد است و خطا هم بالاست ، البته با initial weight های دیگر میتوان خطا را بهتر کرد و شرایط همگرایی بهتر بشود.

۸-۱- نتیجه گیری

بر اساس نتایج به دست آمده از روش های مختلف بازیابی سیگنال پراکنده، مشخص می شود که هر روش دارای نقاط قوت و ضعف خاص خود است.

برای Subset Selection ، کمترین خطای 1.99e-15 و زمان اجرا 0.30 ثانیه را به دست آوردیم. با این حال، این روش با استفاده از نُرم 2 سیگنال پراکنده ای به ما نمی دهد، بنابراین ممکن است بهترین انتخاب برای بازیابی پراکنده نباشد.

استفاده از روش نرم 2 خطای 6.8e-15 و زمان اجرا 0.027 را به ما داد، اما سیگنال پراکنده ای به ما نداد.

برای MP، خطای 2.45 و زمان اجرا 0.0016 را به دست آوردیم، اما خطا از روش های دیگر بیشتر است.

برای OMP، به خطای 2.17 و زمان اجرا 0.0085 رسیدیم که نسبت به MP بهبود یافته است.

با استفاده از برنامه نویسی خطی BP، همان خطای انتخاب زیرمجموعه (subset selection) را با زمان اجرای کوتاهتر 0.00967 به دست آوردیم.

در نهایت، استفاده از IRLS به ما خطای 5.27 با زمان اجرا 0.0139 داد.

در نتیجه، بهترین روش به الزامات و محدودیت های خاص مسئله بستگی دارد. اگر پراکندگی یک نیاز کلیدی است، انتخاب زیر مجموعه و برنامه ریزی خطی BP هر دو گزینه های خوبی هستند که BP سرعت بالاتری دارد .اما اگر سرعت مهم و امکان استفاده از BP نیز نیست ، MP و OMP سریعترین روشها هستند.