سعید غفوری **۹۵۰۷۸۹۴**

گزارش پروژه درس مدل‌های گرافیکی احتمالاتی

موضوع پروژه: پیاده سازی یک مدل سازی از گراف‌های نمایی اتفاقی (ERGM)[1]

مقدمه

یکی از معضلات موجود در کلیه مسائل یادگیری ماشین عدم دسترسی به مجموعه داده[2] کافی است و در بسیاری از موارد نیاز به مطالعه داده‌هایی که توسط یک الگوریتم تولید داده اتفاقی ایجاد شده‌اند می‌تواند مفید واقع شود اما تولید این داده‌ها به صورت کاملا اتفاقی نمی‌تواند به ما شهود دقیقی در آزمایش‌های انجام شده بر روی مساله ما بدهد بنابراین نیاز به تولید داده شبه اتفاقی[3]باشد منظور از داده شبه‌اتفاقی شبه‌اتفاقی داده‌ای است که با وجود این‌که به صورت اتفاقی تولید شده است اما با این حال در بعضی خواص دارای تنظیم می‌باشند مثلا می‌توانیم توزیع آماری داده‌ها یا مثلا تعداد آن‌ها را مشخص کنیم.

بسیاری از کارهای انجام شده در یادگیری ماشین به طور مستقیم و یا غیرمستقیم به بررسی و تحلیل گراف‌ها می‌پردازند و ماهیت داده‌های آن‌ها دارای توصیفی گراف مانند می‌باشد. بنابراین برای تولید مجموعه داده آن‌ها همانطور که گفته شد به دلیل این‌که گرافی که کاملا اتفاقی تولید شده باشد نمی‌تواند اطلاعات مناسبی به ما بدهد نیاز به تولید گراف‌های شبه اتفاقی داریم.

گراف‌های نمایی اتفاقی (ERGM)

یکی از دسته گراف‌های معروف که برای ایجاد گراف‌های اتفاقی با خواصی مدنظر الگوریتم ما استفاده می‌شود گراف‌های نمایی اتفاقی می‌باشد که در ادامه به معرفی آن‌ها میپردازیم.

برای شهود بهتر بدون از دست دادن کلیت مساله فرض میکنیم گرافی که در ادامه تولید می‌کنیم گراف رابطه دوستی در یک شبکه اجتماعی باشد.

اگر در گراف مورد بررسی ما دو گره[4] i و j دارای رابطه باشند این مورد ممکن است ناشی از وجود یک اتصال[5] مستقیم و یا یک ارتباط غیر مستقیم که ناشی از اتصال هردوی این گره‌ها به گره سوم k باشد. اگر بخواهیم در گرافی که قصد ایجاد آن را داریم این ارتباطات را در نظر بگیریم ممکن است با شرایطی رو به رو شویم که در آن تعداد زیادی از گره‌ها با هم در ارتباط باشند زیرا که این رابطه تعدی‌وار دوستی‌ها ممکن است همینطور تا ارتباط بین تمام گره‌ها ادامه پیدا کند. هدف در اینجا این است که علاوه بر در نظر گرفتن صرف تعداد گره‌ها در گراف اتفاقی روابطی از قبیل ارتباطات بین گره‌ها را در نظر بگیریم.

ما در اینجا به عنوان مثال برای گرافی که قصد تولیدش را داریم دو نوع رابطه زیر را بین نودها در نظر می‌گیریم:

- تعداد اتصالات در گراف
- تعداد مثلث‌ها در گراف

---

[1] Exponential Random Graphs
[2] Dataset
[3] Pseudo random
[4] node
[5] link

منظور از مثلث سه گره‌ای است که از طریق یک رابطه تعدی به هم متصل‌اند همان‌طور که در شکل ۱ مشاهده می‌کنید.

این دو معیار را طبق رابطه زیر فرموله‌سازی می‌کنیم:

$$\beta_L \#links(g) + \beta_T \#triangles(g)$$

ما می‌خواهیم که احتمال تشکیل هر گراف مرتبط با این رابطه باشد:

$$\beta_L L(g) + \beta_T T(g)$$

پس قرار می‌دهیم:

$$\Pr(g) \sim \beta_L L(g) + \beta_T T(g)$$

از آن‌جایی که رابطه بالا یک تناسب است و تساوی مستقیم نیست می‌توانیم در سمت راست رابطه $exponential$ مقدار موجود را قرار دهیم. یعنی خواهیم داشت:

$$\Pr(g) \sim \exp[\beta_L L(g) + \beta_T T(g)]$$

طبق قضیه همرسلی-کلیفورد[6] داریم:

هر مدل از گراف‌ها را می‌توان با استفاده از ترکیبی از رابطه‌های آماری بین نودهای آن‌ها بیان کرد.

به طور مثال خانواده گراف‌های اردوش-رینی[7] که رابطه زیر بین گره‌های آن‌ها برقرار است را در نظر بگیرید:

$$P = probabilithy\ of\ a\ link \quad L(g) = number\ of\ links\ in\ g$$

$$\Pr[(g)] = p^{L(g)}(1-p)^{\frac{n(n-1)}{2} - L(g)}$$

می‌توانیم اعمال ریاضی زیر را روی آن انجام دهیم:

$$\Pr[(g)] = p^{L(g)}(1-p)^{\frac{n(n-1)}{2} - L(g)}$$

$$= [\frac{p}{1-p}]^{L(g)}(1-p)^{\frac{n(n-1)}{2}}$$

$$= \exp[\log\left(\frac{p}{1-p}\right)L(g) - \log\left(\frac{1}{1-p}\right)n(n-1)/2]$$

$$= \exp[\beta_1 s_1(g) - c]$$

همان‌طور که مشاهده می‌شود پس از ساده‌سازی موفق شدیم که این دسته از گراف‌ها را به صورت رابطه گفته شده دربیاوریم.

به جهت آن‌که بتوانیم رابطه گفته شده را به صورت احتمالی بنویسیم آن را بر فاکتور نرمال‌سازی زیر که مجوع همان رابطه برای سایر گره‌های گراف است تقسیم می‌کنیم:

---

[6] Hammersly-Cliford

[7] Erdos-Reyni

$$\Pr(g) = \frac{\exp[\beta_L L(g) + \beta_T T(g)]}{\sum_{g'} \exp[\beta_L L(g') + \beta_T T(g')]}$$

که پس از محاسبه $exponential$ داریم:

$$\Pr(g) = \exp[\beta_L L(g) + \beta_T T(g) - c]$$

برای هر نوع از گراف که داشته باشیم با استفاده از تخمین پارامتر از طریق محاسبه $max\ likelihood$ از رابطه بالا و تخمین پارامترهای مربوطه که بنابه رابطه آماری انتخاب شده برای تخمین گراف تعیین می‌شوند توزیع مطلوب که در بهترین حالت بتواند شرایط اولیه روابط آماری مشخص شده توسط ما را ارضا کند را مشخص کنیم. مثلا در رابطه بالا پارامترهایی که نیاز به تخین آن‌ها داریم β_L و β_T می‌باشد. در راه این تخمین می‌توانیم از روش‌های مختلف نمونه برداری مانند نمونه برداری گیبز[8] یا متروپولیس هیستینگ[9] استفاده کنیم.

تعریف پروژه

مراحلی که در پروژه انجام خواهیم داد به این ترتیب خواهد بود:

۱. ابتدا دو تا واحد آماری زیر را برای گراف تعریف خواهیم کرد.
   a. تعداد اتصالات یا همان تعداد یال‌های گراف
   b. تعداد مثلث‌ها
۲. سپس رابطه نمایی که در بالا گفته شد را با توجه به این دو فاکتور حساب میکنیم.
۳. از آن‌جایی که برای محاسبه ضرایب رابطه نیاز به تولید تمام حالات داریم به جای آن از یکی از روش‌های نمونه گیری استفاده خواهیم کرد.
۴. روش نمونه گیری مورد استفاده ما برای نمونه گیری MCMC و نوعی از متروپولیس هیستینگ خواهد بود.
۵. در نهایت با استفاده از نمونه‌های به دست آمده مقدار بهینه پارامترهای گفته شده را به روشی تکراری[10] حساب خواهیم کرد.
۶. در خروجی مقدار بهینه و نمودار تغییرات پارامترها در تکرارهای مختلف الگوریتم را به ازای نمونه‌های تولید شده در خروجی چاپ خواهیم کرد.

ابزار مورد استفاده

زبان برنامه نویسی ما متلب خواهد بود و به جز ابزار متلب برای نمایش گراف‌ها (صرفا نمایش و بیان گراف) از ابزار دیگری به جز ابزار معمول محاسباتی متلب استفاده نخواهد شد.

[8] Gibbs sampling
[9] Metropolis Hasting
[10] iterative

مراحل اجرای برنامه

۱. در ورودی برنامه ابتدا گرافی رندوم با تعداد گره و یال که از کاربر گرفته می‌شود تولید می‌کنیم از این گراف در مراحل بعدی به عنوان سبد اولیه[11] برای تولید گراف‌های اتفاقی مشابه آن در نمونه برداری استفاده خواهد شد.

۲. سپس از کاربر تعداد سمپل‌هایی که برای ضرایب و تعداد گراف‌های اتفاقی‌ای که به منظور امتحان هر ضریب تولید شده می‌خواهیم تولید کنیم گرفته می‌شود.

۳. یک نمونه برای ضرایب به روشی که در ادامه گفته خواهد شد انتخاب می‌کنیم سپس به ازای ضرایب ایجاد شده در این مرحله نمونه‌برداری را بر روی گراف‌ها براساس گراف اولیه‌ای که ساختیم انجام می‌دهیم.

۴. با استفاده از این گراف‌ها و ضرایب محتمل بودن ضرایب فعلی رو تخمین میزنیم.

۵. اگر احتمال این ضرایب نسبت به ضرایب قبلی تخمین زده شده دارای بهبود بود این ضرایب را به ضرایب محتمل اضافه می‌کنیم در غیراینصورت مجددا مقدار ضرایب تولید شده از تکرار قبلی را به ضرایب محتمل اضافه می‌کنیم.

۶. در نهایت در سه آرایه ضرایب مثلث، ضرایب یال‌ها و احتمال به ازای هرکدام از این ضرایب را برمی‌گردانیم.

۷. با پیدا کردن اندیس بزرگترین مقدار احتمال از طریق ماکزیمم‌گیری بر روی آرایه احتمال‌ها، ضرایب بهینه را که در آرایه ضرایب در اندیس متناظر بزرگترین مقدار احتمال قرار دارند را به کاربر برمی‌گردانیم.

توابع برنامه

برنامه متلب نوشته شده شامل توابع زیر می‌باشد:

- randomGraph
- ergmWeights
- randomGraphForMcmc
- mcmc
- fit
- main

که در ادامه به توضیح هرکدام از این توابع خواهیم پرداخت.

تابع randomGrpah

در چندجای این برنامه نیاز به تولید گراف‌های اتفاقی داریم به همین منظور تابعی برای ایجاد این گراف‌ها تولید می‌کنیم. این تابع تعداد گره‌ها و یال‌های گراف اتفاقی که می‌خواهیم تولید کنیم از ورودی می‌گیرد و در خروجی گراف اتفاقی با همین تعداد گره و یال به ما برمی‌گرداند.

[11] Initial seed

main.m | fit.m | mcmc.m | weightSum.m | randomGraphForMcmc.m | ergmWeight.m | vline.m | randomGraph.m

```
function G = randomGraph( nodes , links )
%-----------------generating a random graph

allEdge = nchoosek(1:nodes,2);
allEdge = allEdge';
choosenEdgeIndex = randsample(1:(nodes*(nodes-1))/2,links);
choosenEdge = [];
for i = 1:links
    choosenEdge = [choosenEdge allEdge(:,choosenEdgeIndex(i))];
end
s = choosenEdge(1,:);
t = choosenEdge(2,:);
G = graph(s,t);

end
```

شکل ۱ تابع RandomGraph

برای پیاده سازی گراف از کتابخانه گراف متلب استفاده کردیم و با تولید یال رندوم به تعدادی که لازم داریم گراف را ایجاد می‌کنیم.

تابع ergmWeight

این تابع از رابطه‌ای که در بخش قبل توضیح داده شد استفاده می‌کند و رابطه بین ضریب گراف‌ها را حساب می‌کند.

main.m | fit.m | mcmc.m | weightSum.m | randomGraphForMcmc.m | ergmWeight.m | vline.m | randomGraph.m

```
function weight = ergmWeight(G,edgeC,triC)
%   edgeC is edge coefficient
%   triC is triangle coefficient
%   this funcion compute the each function by the
%   exponential function

%   extracting nummber of edges in the graph
numOfEdges = numedges(G);
%   find the number of triangles in the graph
adjacencyMatrix = full(adjacency(G));
adjacencyP3 = adjacencyMatrix^3;
numOfTri = trace(adjacencyP3)/6;
%   calculating the weight through the formula
weight = exp(numOfEdges * edgeC + numOfTri * triC);
end
```

شکل ۲ تابع ergmWeight

تعداد یال‌ها را مستقیما از گرافی که بدست آورده‌ایم محاسبه می‌کنیم و تعداد مثلث‌ها را هم براساس رابطه‌ای که از ماتریس مجاورت گراف بدست می‌آید محاسبه می‌کنیم. ضرایب مثلث‌ها و یال‌ها را هم که در ورودی‌ها به تابع داده‌ایم.

تابع mcmc و randomGraphForMcmc

در این تابع قصد داریم با یک پیاده سازی ساده از یکی از روش‌های نمونه‌برداری مونت کارلو به نام متروپولیس هیستینگ تعدادی نمونه نزدیک به گراف اولیه‌ای که به برنامه دادیم (می‌توانستیم گراف را دستی به برنامه بدهیم اما به جهت راحتی کار و تست از یک گراف که از طریق تابع randomGraph ایجاد کرده‌ایم استفاده می‌کنیم.) نحوه کار بدین‌گونه است که ابتدا از تابع mcmc شروع می‌کنیم و به آن گراف اولیه‌مان را می‌دهیم همچنین در ورودی باید پارامترهای ضریب مثلث و یال‌ها را وارد می‌کنیم سپس گرافی اتفاقی با استفاده از تابع randomGraph با اندازه داده شده تشکیل می‌دهیم و سپس در یک while تا زمانی که دو شرط که برای نمونه برداری هستند یال کم و زیاد می‌کنیم و این یال کم و زیاد کردن به طور اتفاقی در تابعی به نام randomGraphForMcmc انجام خواهد شد.

main.m | fit.m | mcmc.m | weightSum.m | randomGraphForMcmc.m | ergmWeight.m | vline.m | randomGraph.m

```
function mcmcGraphs = mcmc(seedG , edgeC , triC , numOfSample )
%   Monte Carlo sampling
%   seedG: seed graph
%   edgeC: edge coefficient
%   triC: triangle coefficient
%   numOfSample: number of sample we wish to generate
%   mcmcGraphs: cell array of generated graph
mcmcGraphs = [];
%   count the number nodes
adjacencyMatrix = full(adjacency(seedG));
[numOfNodes , 1] = size(adjacencyMatrix);
% make a new random graph at size of seed and a random number of edges
numOfEdges = randi((numOfNodes * (numOfNodes - 1))/2);
currentGraph = randomGraph(numOfNodes , numOfEdges);
currentW = ergmWeight(currentGraph , edgeC , triC);
i = 1;
while i<=numOfSample
    generatedGraph = randomGraphForMcmc(currentGraph);
    generatedGraphW = ergmWeight(generatedGraph , edgeC , triC);
    if generatedGraphW > currentW || rand < generatedGraphW / currentW
        mcmcGraphs = [mcmcGraphs {generatedGraph}];
        currentW = generatedGraphW;
    else
        i = i - 1;
    end
    i = i + 1;
```

شکل ۳ تابع mcmc

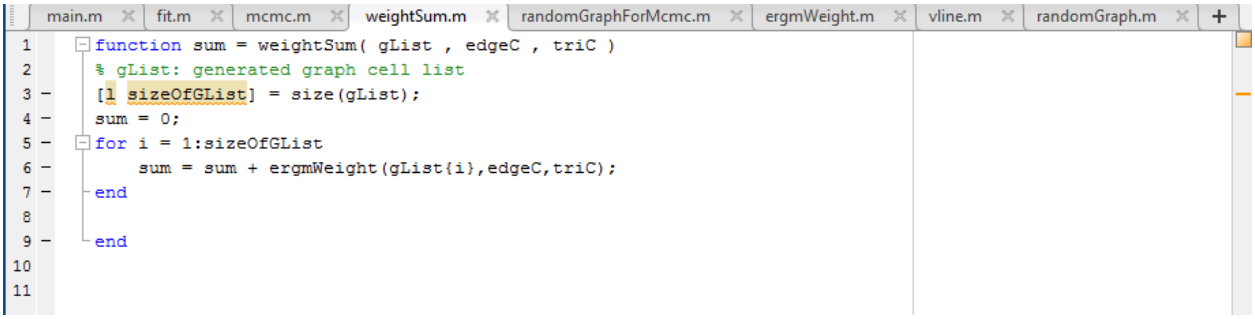
main.m | fit.m | mcmc.m | weightSum.m | randomGraphForMcmc.m | ergmWeight.m | vline.m | randomGraph.m

```
function GM = randomGraphForMcmc( G )
%   a graph with randomly an edge deleted or added to it
%   GM: modefied graph
%   complement of a matrix
adjacencyMatrix = full(adjacency(G));
[numOfNodes , 1] = size(adjacencyMatrix);
cAdjacencyMatrix = xor(adjacencyMatrix , ones(numOfNodes , numOfNodes));
cAdjacencyMatrix = double(cAdjacencyMatrix & xor(diag(ones(1,numOfNodes)),ones(numOfNodes,numOfNodes)))
cGraph = graph(cAdjacencyMatrix);
cEdgesOfGraph = table2array(cGraph.Edges)';
[1 cNumOfEdges] = size(cEdgesOfGraph);
%   find the graph edges and number of them
edgesOfGraph = table2array(G.Edges)';
[1 numOfEdges] = size(edgesOfGraph);
%   50 50 chance of choosing whether add or remove an edge
addOrRemove = rand;
%   remove an edge
if numOfEdges == 0 || addOrRemove < 0.5 || numOfNodes ~= (numOfNodes * (numOfNodes-1))/2
    REdgeIndex = randi(numOfEdges); %   to be removed edge index
    GM = rmedge(G , edgesOfGraph(1,REdgeIndex) , edgesOfGraph(2,REdgeIndex));
%   add an edge
else
    REdgeIndex = randi(cNumOfEdges); %   to be removed edge index
    GM = addedge(G , cEdgesOfGraph(1,REdgeIndex) , cEdgesOfGraph(2,REdgeIndex));
end
```

شکل ۴ تابع randomGraphForMcmc

تابع weightSum

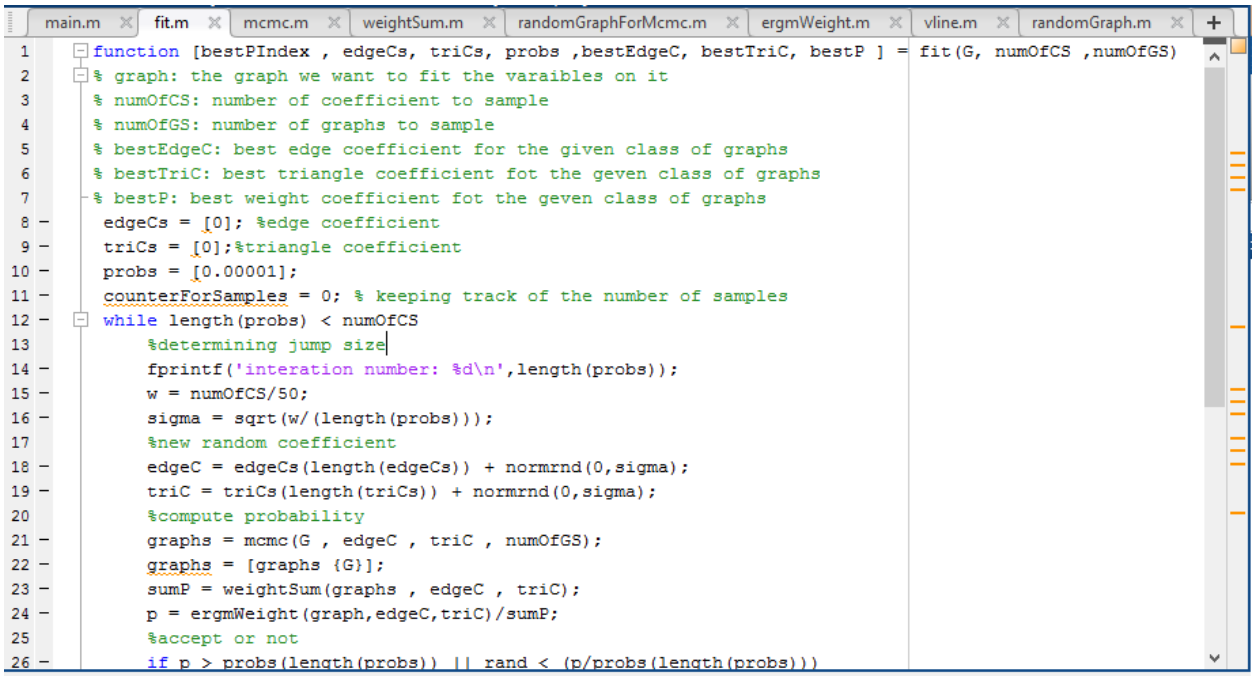
این تابع برای محاسبه وزن (مجموع احتمالات) مجموعه گراف های تولید شده از نمونه‌برداری انجام شده با توابع mcmc و randomGraphMcmc به کار می‌رود.



شکل ۵ تابع *weightSum*

تابع fit

تابع اصلی برنامه می‌باشد این تابع وظیفه دارد که با نمونه برداری از مقادیر ضرایب یال‌ها و مثلث‌ها ضرایب را تخمین بزند. در هر مرحله از تکرار الگوریتم با محاسبه نسبت قبولی[۱۲] رد و یا مورد پذیرش بودن نمونه جاری را بررسی می‌کنیم این امر تا زمانی انجام می‌گیرد تا تعداد نمونه از ضرایب درخواست شده توسط کاربر تولید شود قابل قبول بودن این نمونه‌ها توسط مجموعه گراف‌هایی که از مرحله نمونه برداری گراف‌ها ایجاد میشود تایید می‌شود. ضمنا نمونه برداری ما از روی توزیع نرمال انجام خواهد شد. هدف ما این است که هرچه به جواب نزدیک‌تر می‌شویم گام‌های کوچک‌تری برداریم.



شکل ۶ تابع *fit* بخش اول

[12] Acceptance ratio

main.m | fit.m | mcmc.m | weightSum.m | randomGraphForMcmc.m | ergmWeight.m | vline.m | randomGraph.m

```
19 -        triC = triCs(length(triCs)) + normrnd(0,sigma);
20          %compute probability
21 -        graphs = mcmc(G , edgeC , triC , numOfGS);
22 -        graphs = [graphs {G}];
23 -        sumP = weightSum(graphs , edgeC , triC);
24 -        p = ergmWeight(graph,edgeC,triC)/sumP;
25          %accept or not
26 -        if p > probs(length(probs)) || rand < (p/probs(length(probs)))
27 -           edgeCs = [edgeCs edgeC];
28 -           triCs = [triCs triC];
29 -           probs = [probs p];
30 -        else
31 -            edgeCs = [edgeCs edgeCs(length(edgeCs))];
32 -            triCs = [triCs triCs(length(triCs))];
33 -            probs = [probs probs(length(probs))];
34 -        end
35 -    end
36
37 -    [l bestPIndex] = max(probs);
38 -    bestEdgeC = edgeCs(bestPIndex);
39 -    bestTriC = triCs(bestPIndex);
40 -    bestP = probs(bestPIndex);
41
42 -  end
```

شکل ۷ تابع fit بخش دوم

تابع main

تابع اصلی برنامه است که در آن ابتدا یک گراف اتفاقی برای گراف پایه برنامه تشکیل می‌دهیم و در نهایت با صدا زدن توابعی که گفته شد برنامه مقادیر بهینه ضرایب یاد شده را محاسبه خواهد کرد. و سپس مقدار بهینه را در خروجی چاپ خواهد کرد.

main.m | fit.m | mcmc.m | weightSum.m | randomGraphForMcmc.m | ergmWeight.m | vline.m | randomGraph.m

```
1 -   clear;
2 -   close;
3     %initial random graph statistics
4 -   fprintf('--------graph statistics--------\n');
5 -   nodes = input('enter the number of nodes:\n');
6 -   links = input('enter the number of linkes:\n');
7     % algorithm parameters
8 -   fprintf('--------algorithm parameters--------\n');
9 -   numOfCS = input('enter the number of coefficient samples:\n');
10 -  numOfGS = input('enter the number of graph samples per interation:\n');
11    %generate first graph as the seed
12 -  G = randomGraph( nodes , links );
13 -  [bestPIndex, edgeCs, triCs, probs ,bestEdgeC, bestTriC, bestP ] = fit(G, 50 ,50);
14 -  x = 1:length(probs);
15 -  subplot(1,2,1);
16 -  plot(x, edgeCs, x, triCs);
17 -  hold;
18 -  h = vline(bestPIndex,'g','optimal');
19 -  legend('edge coefficient','triangle coefficient','Location','northeast');
20 -  subplot(1,2,2);
21 -  plot(G);
22 -  fprintf('the optimal edge coefficient for current family of graphs is: %d\n',bestEdgeC);
23 -  fprintf('the optimal triangle coefficient for current family of graphs is: %d\n',bestTriC);
24
```

شکل ۸ تابع main

نمونه از اجرای برنامه

در این بخش نمونه‌ای از اجرای برنامه را خدمتتان نشان خواهیم داد.

در ابتدا اسکریپت main که نقطه شروع برنامه است را صدا می‌زنیم.

```
Command Window
>> main
```

سپس برنامه از ما می‌خواهد که مشخصات گراف اولیه که به عنوان سبد اولیه می‌دهیم را وارد کنیم همان‌طور که در قبل هم گفته شد این گراف به صورت اتفاقی تولید خواهد شد و فقط تعداد گره‌ها و یال‌ها از سوی کاربر قابل تنظیم است.

```
Command Window
>> main
--------graph statistics--------
enter the number of nodes:
```

که ما هم به‌طور مثال از برنامه می‌خواهیم گرافی با ۸ گره و ۱۴ یال ایجاد کند.

```
Command Window
>> main
--------graph statistics--------
enter the number of nodes:
8
enter the number of linkes:
14
```

در مرحله بعدی پارامترهای مورد نیاز الگوریتم را به آن می‌دهیم.

```
Command Window
>> main
--------graph statistics--------
enter the number of nodes:
8
enter the number of linkes:
14
--------algorithm parameters--------
enter the number of coefficient samples:
```

تعداد نمونه‌های ضرایب را ۵۰ و تعداد نمونه‌های گراف تولید شده به ازای هر نمونه از ضرایب می‌باشد.

```
Command Window
>> main
--------graph statistics--------
enter the number of nodes:
8
enter the number of linkes:
14
--------algorithm parameters--------
enter the number of coefficient samples:
50
enter the number of graph samples per interation:
50
```

برنامه شروع به اجرا می‌کند و وارد هر تکراری که می‌شود آن‌را در خروجی نشان می‌دهد.

```
Command Window
--------graph statistics--------
enter the number of nodes:
8
enter the number of linkes:
14
--------algorithm parameters--------
enter the number of coefficient samples:
50
enter the number of graph samples per interation:
50
interation number: 1
interation number: 2
interation number: 3
interation number: 4
interation number: 5
interation number: 6
interation number: 7
interation number: 8
interation number: 9
```

و در نهایت پس از اتمام اجرا در خروجی برنامه مقدار بهینه تخمین زده شده برای ضرایب مثلث‌ها و یال‌ها را به نشان خواهد داد.

```
Command Window
interation number: 34
interation number: 35
interation number: 36
interation number: 37
interation number: 38
interation number: 39
interation number: 40
interation number: 41
interation number: 42
interation number: 43
interation number: 44
interation number: 45
interation number: 46
interation number: 47
interation number: 48
interation number: 49
Current plot held
the optimal edge coefficient for current family of graphs is: -2.026115e+00
the optimal triangle coefficient for current family of graphs is: 1.711848e+00
>>
```

همچنین برای ما در خروجی و در سمت راست گراف اولیه مورد استفاده و در سمت چپ نمودار تغییرات ضرایب را در طول تکرارهای متوالی برنامه به ما نشان می‌دهد. خط سبز در شکل سمت راست نشان دهنده ضرایبی است که به ازای آن مقدار احتمال ماکزیمم می‌شود.