« پروژه پایان نامه »

عنوان : طراحی نرم افزار پخش بار به روش نیوتن رافسون به

زبان C++ Builder

استاد راهنما : مهندس جوان

دانشجو : محمدرضا راژ

دانشگاه آزاد اسلامی واحد تهران جنوب

فهرست صفحه

۴- الگوریتم مربوط به وارون ژاکوبین

۵- الگوریتم مربطو به محاسبه Q_{cali}, P_{cali}

۶- الگوریتم مربوط به محاسبه ماتریس $\Delta Q_i, \Delta P_i$

۷-الگوریتم مربوط به ضرب وارون ژاکوبین در ماتریس $\Delta Q_i, \Delta P_i$

۸- الگوریتم مربوط به محاسبه $|V_i|_{new}, S_{newi}$

۹- الگوریتم تست شرط

۱۰- الگوریتم مربوط به چاپ جوابهای مسئله در خروجی

فصل پنجم – مروری بر دستورات برنامه نویسی C++

۱- انواع داده

۲- متغیرها

۳۳- تعریف متغیر

۴- مقدار دادن به متغیر

۵- عملگرها

۶- عملگرهای محاسباتی

۷- عملگرهای رابطه‌ای

۸- عملگرهای منطقی

۹- عملگر Sizcof

۱۰- ساختار تکرار for

با سپاس به درگاه حضرت حق ، که دانش را باب معرفت و معرفت را باب کمال آفرید. می‌ستایم او را که بهترین دانایان است. می‌خوانم او را که تنها یاری دهنده بندگان است. یگانه خالقی که چنان می‌آفریند که ادراک از درکش عاجز می‌ماند. پروردگارا تو را سپاس می‌گویم که زره‌ای از بی‌کران دانش خود را بر من نمودی ، تا در این راه تو را بهتر بشناسم. امید آن دارم که همواره تنها یاورم باشی و درهای جدیدی از معرفت خویش را بر من باز نمایی. سپاس و قدردانی می‌کنم از پدر و مادر عزیزم که سالها مرا یاری کردند و همواره مشوق من به راه دانستن بودند. و تشکر می‌کنم از همه استاتیدی که دانش و تجربه سالهای عمر خود را برای من هزینه کردند تا بدانم آنچرا که نمی‌دانم. امیدوارم همواره بتوان قدردان زحمات همه کسانی که مرا یاری کردند باشم. ان‌شاءالله.

تابستان ۸۵

محمد رضا راژ

مقدمه :

بی شک صنعت برق مهمترین و حساسترین صنایع در هر کشور محسوب می‌شود. بطوریکه عملکرد نادرست تولید کننده‌ها و سیستم‌های قدرت موجب فلج شدن ساختار صنعتی ، اقتصادی ، اجتماعی و حتی سپاسی در آن جامعه خواهد شد. از زمانیکه برق کشف و تجهیزات برقی اختراع شدند. تکنولوژی با سرعت تساعدی در جهت پیشرفت شتاب گرفت. بطوریکه می‌توان گفت در حدود دویست سال اخیر نود درصد از پیشرفت جامع بشری به وقوع پیوست. و شاید روزی یا هفته‌ای نباشد که دانشمندان سراسر جهان مطلب جدیدی در یکی از گراشیهای علم برق کشف و عنوان نکنند. و انسان قرن بیست و یکم بخش قابل توجه‌ای از آسایش رفاه خود را مدیون حرکت الکترونها می‌باشد. و دانشمندان در این عرصه انسانهای سختکوش بودند که همه تلاش خود را برای افراد راحت طلب بکار بستند.

در آغاز شکل گیری شبکه‌های برقی ، مولدها ، برق را بصورت جریان مستقیم تولید می‌کردند و در مساحتهای محدود و کوچک از آنها بهره‌مند می‌شد. و این شبکه‌ها بصورت کوچک و محدود استفاده می‌شد. با افزایش تقاضا در زمینه استفاده از انرژی الکتریکی دیگر این شبکه‌های کوچک پاسخگوی نیاز مصرف کننده‌ها نبود و می‌بایست سیستم‌های برقرسانی مساحت بیشتری را تحت پوشش خود قرار می‌دادند. از طرفی برای تولید نیز محدودیتهایی موجود بود که اجازه تولید انرژی الکتریکی را در هر نقطه دلخواه به مهندسین برق نمی‌داد. زیرا که نیروگاه‌ها می‌بایست در محلهایی احداث می‌شد که انرژی بطور طبیعی یافت می‌شد. انرژیهای طبیعی مثل : آب ، باد ، ذغال سنگ وغیره بنابراین نیروگاه‌ها را می‌بایست در جاهایی احداث می‌کردند که یا در آنجا آب و یا باد و یا

ذغال سنگ و دیگر انرژیهای سوختی موجود بود. بدین ترتیب نظریه انتقال انرژی الکتریکی از محل تولید انرژی تا محل مصرف پیش آمد. این انتقال نیز توسط برق جریان مستقیم امکان‌پذیر نبود. زیرا ولتاژ در طول خط انتقال افت می کرد و در محل مصرف دیگر عملاً ولتاژی باقی نمی‌ماند. بنابراین مهندسین صنعت برق تصمیم گرفتند که انرژی الکتریکی را بطور AC تولید کنند تا قابلیت انتقال داشته باشد. و این عمل را نیز توسط ترانسفورماتورها انجام دادند. ترانسفورماتورها می‌توانستند ولتاژ را تا اندازه قابل ملاحظه‌ای بالا برده و امکان انتقال را فراهم آورند. مزیت دیگری که ترانسفورماتورها به سیستم‌های قدرت بخشیدند. این بود که با بالا بردن سطح ولتاژ ، به همان نسبت نیز جریان را پائین می آوردند ، بدین ترتیب سطح مقطع هادیهای خطوط انتقال کمتر می‌شد و بطور کلی می‌توانستیم کلیه تجهیزات را به وسیله جریان پائین سایز نماییم. و این امر نیز از دیدگاه اقتصادی بسیار قابل توجه می‌نمود.

بدین ترتیب شبکه‌های قدرت AC شکل گرفت و خطوط انتقال و پستهای متعددی نیز برای انتقال انرژی الکتریکی در نظر گرفته شد. و برای تأمین پیوسته انرژی این شبکه‌ها به یکدیگر متصل شدند و تا امروه نیز در حال گسترش و توسعه می‌باشند. هرچه سیستمهای قدر الکتریکی بزرگتر می‌شد بحث بهره‌برداری و پایداری سیستم نیز پیچیده‌تر نشان می‌داد. و در این راستا مراکز کنترل و بهره بردار از سیستم‌های قدرت می‌بایست در هر لحظه از ولتاژها و توانهای تمامی پست‌ها و توانهای جاری شده در خطوط انتقال آگاهی می‌یافتند. تا بتوانند انرژی را بطور استاندارد و سالم تا محل مصرف انتقال و سپس توزیع کنند. این امر مستلزم حل معادلاتی بود که تعداد مجهولات از تعداد معلومات بیشتر بود. حل معادلاتی که مجهولات بیشتری از معلومات آن دارد نیز فقط در

فضای ریاضیاتی با محاسبات عدد امکان‌پذیر است که در تکرارهای مکرر قابل دستیابی است. در صنعت برق تعیین ولتاژها و زوایای ولتاژها و توانهای اکتیو و راکتیو در پستها و نیروگاهها را با عنوان پخش بار (load flow) مطرح می‌شود.

پخش بار در سیستمهای قدرت دارای روشهای متنوعی می‌باشد که عبارتند از : روش نیوتن ۰ رافسون ، روش گوس – سایدل ، روش Decaupled load flow و روش Fast decaupled load flow که هر یک دارای مزیت‌های خاص خود می‌باشد. روش نیوتن- رافسون یک روش دقیق با تکرارهای کم می‌باشد که جوابها زود همگرا می‌شود ، اما دارای محاسبات مشکلی است. روش گوس – سایدل دقت کمتری نسبت به نیوتن رافسون دارد و تعداد و تکرارها نیز بیشتر است اما محاسبات ساده‌تری دارد. روش Decaupled load flow یک روش تقریبی در محاسبات پخش بار است و دارای سرعت بالایی می‌باشد ، و زمانی که نیاز به پیدا کردن توان اکتیو انتقالی خط مطرح است مورد استفاده می‌باشد. روش Fast decaupled load flow نیز یک روش تقریبی است که از سرعت بالایی نیست به نیوتن رافسون و گوس سایدل برخوردار می‌باشد. و از روش Decaupled load flow نیز دقیق‌تر می‌باشد. اما مورد بحث این پایان‌نامه روش نیوتن – رافسون است که در ادامه به آن می‌پردازیم.

فصل اول

شرحی بر پخش بار

پخش بار :

جهت طراحی و توسعه آتی و بهترین عملکرد یک سیستم قدرت پخش بار ابزار توانمندی برای دستیابی به این مهم می‌باشد. اطلاعاتی که پس از انجام محاسبات پخش بار در شبکه در اختیار قرار می‌گیرد شامل ولتاژ ، زاویه ولتاژ ، توان اکتیو و راکتیو تمامی شین‌ها و همچنین توانهای اکتیو و راکتیو جاری در خطوط می‌باشد. مضافاً به این که اطلاعات بیشتری توسط نرم افزارهایی که شرکتهای برق منطقه ای از آن سود می‌برند در اختیار قرار می‌گیرد.

با توجه به راه حلهای طولانی و تکرارهای پیاپی این روشها و همچنین فرصت ناچیزی که مهندسین برق در حین به وقوع پیوستن اتفاقات اجتناب ناپذیر و غیر قابل پیش بینی در اختیار دارند ، جایگاه یک نرم افزار قدرتمند در طراحی و عملکرد مناسب سیستم‌های قدرت که از ارزش اقتصادی بسیار زیادی نیز برخوردارند مشخص می‌شود.

این نرم افزارها با توجه به در اختیار بودن پردازنده‌های پر سرعت نسل جدید می‌توانند تمامی اطلاعات مربوط به شبکه مورد مطالعه را به سرعت در اختیار قرار دهند. مطالعات پخش توان استخوانبندی اصلی تجزیه و تحلیل و طراحی سیستم قدرت را تشکیل می‌دهد. این مطالعات دارای کاربردها و مزیتهای بسیاری می‌باشد که شامل : برنامه‌ریزی ، بهره‌برداری ، برنامه ریزی زمانبندی اقتصادی ، و تبادل توان بین شرکتهای برق منطقه‌ای می‌باشد. مضافاً به این که این مطالعات برای تحلیل‌های دیگری همچون مطالعات پایداری گذرا ، خطاهای احتمالی ایجاد شونده در شبکه‌ها ، طراحی پست ، طراحی خط انتقال و حوادث و رخدادهای غیر مترقبه در سیستم‌های قدرت مورد نیاز است. وقتی بخواهیم پست جدیدی احداث نماییم. برای انتخاب کلید مناسب در

آن پست می‌بایست ابتدا محاسبات اتصال کوتاه را انجام داد و جریان اتصال کوتاه را برای شین آن پست در نظر گرفت و سپس کلید مورد نظر را انتخاب نمود. بنابراین برای اینکه بتوانیم محاسبات اتصال کوتاه را انجام دهیم می‌بایست جواب حاصل از مطالعات پخش بار را در اختیار داشته باشیم. و یا هر گاه برحسب احتمال نیروگاهی از شبکه سراسری از مدار خارج شود. برای بررسی پایداری در شبکه باید به سرعت محاسبات پخش بار را انجام داد تا از پایدار بودن سیستم و یا اینکه آیا دیگر نیروگاه‌ها قادر به تأمین انرژی لازم مصرف کنندگان هستند یا خیر اطمینان حاصل کرد. چرا که در غیر اینصورت ممکن است انرژی مصرفی در سیستم در آن لحظه بیشتر از انرژی تأمین شده توسط ژنراتورهای نیروگاه‌ها باشد که این امر موجب می‌شود تا کل شبکه از دست برود. و همچنین دیگر کاربردهای متنوع محاسبات پخش بار در شبکه‌های برق‌رسانی و مباحث مختلف مهندسی برق اهمیت آنرا آشکار می‌سازد.

برای انجام مطالعات پخش بار می‌بایست ماتریس ادمیتانس شبکه و یا Ybus شبکه و اطلاعات مربوط به شین ها در اختیار باشد. شین‌ها انواع مختلفی دارند که هر یک با توجه به نوع آن شامل اطلاعات خاص خود را می‌باشند. و نوع شین معلومات و مجهولات مربوط به آن شین را معین می‌کند. بطور کی در شبکه‌های قدرت می‌توان پنج نوع شین را در نظر گرفت که شامل موارد زیر است :

۱) شین مرجع یا شناور (Slack or swing)

۲) شین بار (load bus)

۳) شین ولتاژ کنترل نشده (Control bus)

۴) شین نیروگاهی (generator bus)

۵) شین انتقال (ttranse bus)

در زیر به هر یک از موارد فوق پرداخته می‌شود. و معلومات و مجهولات مربوط به آنها مشخص خواهد شد.

شین مرجع یا شناور (Slack or swing bus)

این شین در واقع یک شین نیروگاهی می‌باشد ، که بعنوان مرجع تعیین می شود. به این معنی که ولتاژ این شین را یک و زاویه ولتاژ آنرا صفر در نظر می‌گیریم و مقادیر ولتاژ و زاویه ولتاژ شین‌های دیگر سیستم مورد مطالعه را نسبت به آن سنجش می‌کنیم.

لازم به ذکر است که مرجع بودن این شین با گره مرجع در تحلیل مدارهای الکتریکی به روش تحلیل‌گره تفاوت می‌کند. در واقع در هر سیستمی باید یک شین را بعنوان مرجع انتخاب نمود. این شین حتماً می‌بایست نیروگاهی باشد و معمولاً شین شماره یک را مرجع در نظر می‌گیریم. شین مرجع دارای قدرت توان گیری و توان دهی سریعتری نسبت به شین‌های دیگر می‌باشد.

همانطور که عنوان شد ولتاژ را برای شین مرجع یک و زاویه ولتاژ را صفر در نظر می‌گیریم بنابراین ولتاژ و زاویه آن برای شین مرجع جزء معلومات محسوب می‌شود و توان اکتیو و راکتیو تولیدی در این شین مجهول است. پس از Converge کردن سیستم مقادیر توانهای اکتیو و راکتیو تولیدی در این شین معین می‌شوند.

شین بار (load bus)

شین بار به شینی اطلاق می‌شود که در آن باری برای مصرف قرار گرفته باشد.

و هیچ گونه جبرانسازی در آن صورت نگرفته باشد. به این شین ، شین PQ نیز گفته می‌شود زیرا معلومات ما در آن توان اکتیو و راکتیو مصرفی می‌باشد. ولتاژ و زاویه ولتاژ برای شین بار مجهولند ، و در فرایند محاسبات به دست می‌آیند.

شین ولتاژ کنترل شده (Control bus)

در پستهایی که برای محدود کردن رگولاسیون ولتاژ در یک بازه معین جبرانسازی شده باشد. به آن پست ، شین ولتاژ کنترل شده گویند. هرگاه بر اثر مصرف بارهای سلفی افت ولتاژ حاصل شود ، برای جبراین این افت خازن‌های جبرانساز را وارد مدار کرده و بدین ترتیب ولتاژ را به حد مطلوب شیفت می‌دهند. و بطور عکس هرگاه بر اثر خاصیت خازنی خطوط و مصرف کم افزایش ولتاژ مشاهده شود ، تحت چنین شرایطی راکتورهای جبرانساز به شین متصل می‌شود. در این شین تنها اندازه ولتاژ معلوم بوده و زاویه و توان راکتیور ، سلف یا خازن ، مجهولات را شامل می‌شوند.

شین نیروگاهی (generator bus)

همانطور که از اسم آن پیداست ، این شین یک نیروگاه می‌باشد. به این شین PV نیز می‌گویند ، چرا که ، اندازه ولتاژ و همچنین توان اکتیو تولیدی در آن مشخص و جزء معلومات شین محسوب می‌شود. زاویه ولتاژ و توان راکتیو تولیدی آن از مجهولات شبکه است. و توان راکتیو آنرا بعد از Converge کردن سیستم بدست می آوریم.

شین انتقال (transe bus)

این نوع شین به شینی اطلاق می‌شود که نه به آن باری متصل است و نه جبرانسازی صورت گرفته و نه نیروگاهی در آن قرار دارد. این باس تنها شامل خطوطی می‌باشد که به آن وارد و یا از آن

خارج می‌شوند. بنابراین هیچگونه توان اکتیو و راکتیوی نه در آن تولید و نه در آن مصرف می‌شود. بنابراین این باس معلوماتی را شامل نمی‌شود. بلکه اندازه و زاویه ولتاژ در آن مجهولند. این باس مثل پست‌های کلیدزنی می باشد که در شبکه‌ها موجودند.

فصل دوم

محاسبات ریاضی نرم افزار

حل معادلات جبری غیر خطی به روش نیوتن – رافسون :

رایج‌ترین روش حل معادلات جبری غیر خطی همزمان ، روش نیوتن ، رافسون است. روش نیوتن رافسون یک روش تقریب متوالی است که بر اساس تخمین اولیه مقادیر مجهول و استفاده از بسط سری تیلور بنا شده است. ابتدا به ساکن فرض می‌شود که حل معادله یک بعدی زیر مد نظر باشد.

$$f(x)=C$$

حال اگر $x^{(\circ)}$ تخمین اولیه از حل مسأله بوده و $\Delta x^{(\circ)}$ انحراف کوچکی نسبت به پاسخ صحیح باشد ، داریم :

$$f\left(x^{(o)}+\Delta x^{(o)}\right)=C$$

با بسط سری تیلور سمت چپ معادله بالا در اطراف $x^{(o)}$ خواهیم داشت :

$$f\left(x^{(o)}\right)+\left(\frac{df}{dx}\right)^{(o)}\Delta x^{(o)}+\frac{1}{21}\left(\frac{d^2f}{dx^2}\right)^{(o)}\left(\Delta x^{(o)}\right)+\ldots\ldots\ldots=C$$

با فرض اینکه $\Delta x^{(o)}$ خیلی کوچک باشد ، می‌توان از جملات مرتبه‌های بالاتر چشم پوشی کرد ، که در این صورت می‌توان نوشت :

$$\Delta C^{(o)}\simeq\left(\frac{df}{dx}\right)^{(o)}\Delta x^{(o)}$$

که در آن :

$$\Delta C^{(o)}=C-f\left(x^{(o)}\right)$$

با افزودن $\Delta x^{(o)}$ به مقدار تخمین اولیه ، تقریب دوم بدست می آید :

$$x^{(1)} = x^{(o)} + \frac{\Delta C^{(o)}}{\left(\frac{df}{dx}\right)^{(o)}}$$

و بدین ترتیب کاربرد متوالی این روش ، الگوریتم نیوتن – رافسون را در اختیار قرار می‌دهد :

$$\Delta C^{(k)} = C - f\left(x^{(k)}\right)$$

$$\Delta x^{(k)} = \frac{\Delta C^{(k)}}{\left(\frac{df}{dx}\right)^{(k)}}$$

که در آن k ، تعداد تکرارها می‌باشد ، و همچنین جوابهای معادله به قرار زیر است :

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$$

و همچنین معادلات فوق را می‌توان بصورت زیر مرتب کرد :

$$\Delta C^{(k)} = J^{(k)}.\Delta x^{(k)}$$

که در آن $J^{(k)}$ بصورت زیر تعریف می شود :

$$J^{(k)} = \left(\frac{df}{dx}\right)^{(k)}$$

رابطه فوق نشان می‌دهد که معادله غیر خطی $f(x) - C = 0$ با خط مماس بر منحنی در نقطه $x^{(k)}$ تقریب زده می‌شود. بنابراین یک معادله خطی برحسب تغییرات کوچک متغیرها بدست می‌آید. از تلاقی خط مماس با محوط x مقدار $x^{(k+1)}$ بدست می آید. این نظریه بصورت ترسیمی در شکل زیر نشان داده شده است.

حال اگر معادلات n بعدی زیر را در نظر بگیریم :

$$f_1(x_1, x_2, ...x_n) = C_1$$
$$f_2(x_1, x_2, ...x_n) = C_2$$
$$\vdots$$
$$f_n(x_1, x_2, ...x_n) = C_n$$

بسط سمت چپ معالات بالا بصورت سری تیلور در اطراف تخمین‌های اولیه و چشم پوشی از جملات مرتبه بالاتر ، منجر به روابط زیر می‌شود :

$$(f_1)^{(o)} + \left(\frac{\partial f_1}{\partial x_1}\right)^{(o)} \Delta x_1{}^{(o)} + \left(\frac{\partial f_1}{\partial x_2}\right)^{(o)} \Delta x_2{}^{(o)} + ... + \left(\frac{\partial f_1}{\partial x_n}\right)^{(o)} \Delta x_n{}^{(o)} = C_1$$

$$(f_2)^{(o)} + \left(\frac{\partial f_2}{\partial x_1}\right)^{(o)} \Delta x_1{}^{(o)} + \left(\frac{\partial f_2}{\partial x_2}\right)^{(o)} \Delta x_2{}^{(o)} + ... + \left(\frac{\partial f_2}{\partial x_n}\right)^{(o)} \Delta x_n{}^{(o)} = C_2$$

$$(f_n)^{(o)} + \left(\frac{\partial f_n}{\partial x_1}\right)^{(o)} \Delta x_1{}^{(o)} + \left(\frac{\partial f_n}{\partial x_2}\right)^{(o)} \Delta x_2{}^{(o)} + ... + \left(\frac{\partial f_n}{\partial x_n}\right)^{(o)} \Delta x_n{}^{(o)} = C_n$$

و یا بصورت ماتریسی خواهیم داشت :

$$\begin{bmatrix} C_1 - (f_1)^{(o)} \\ C_2 - (f_2)^{(o)} \\ \vdots \\ C_n - (f_n)^{(o)} \end{bmatrix} = \begin{bmatrix} \left(\frac{\partial f_1}{\partial n_1}\right)^{(o)} & \left(\frac{\partial f_1}{\partial x_2}\right)^{(o)} & ..... & \left(\frac{\partial f_1}{\partial x_n}\right)^{(o)} \\ \left(\frac{\partial f_2}{\partial x_1}\right)^{(o)} & \left(\frac{\partial f_2}{\partial x_2}\right)^{(o)} & ..... & \left(\frac{\partial f_2}{\partial x_n}\right)^{(o)} \\ \vdots & \vdots & & \vdots \\ \left(\frac{\partial f_n}{\partial x_1}\right)^{(o)} & \left(\frac{\partial f_n}{\partial x_2}\right)^{(o)} & ..... & \left(\frac{\partial f_n}{\partial x_n}\right)^{(o)} \end{bmatrix} \begin{bmatrix} \Delta x_1{}^{(o)} \\ \Delta x_2{}^{(o)} \\ \vdots \\ \Delta x_n{}^{(o)} \end{bmatrix}$$

این معادله بصورت زیر خلاصه می‌شود :

$$\Delta C^{(k)} = J^{(k)} \Delta x^{(k)}$$

بنابراین خواهیم داشت :

$$\Delta x^{(k)} = \left[J^{(k)}\right]^{-1} \Delta C^{(k)}$$

و الگوریتم نیوتن – رافسون برای حالت n بعدی بصورت زیر در می‌آید :

$$X^{(k+1)} = X^{(k)} + \Delta X^{(k)}$$

که در آن :

$$\Delta X^{(k)} = \begin{bmatrix} \Delta x_1^{(k)} \\ \Delta x_2^{(k)} \\ \vdots \\ \Delta x_n^{(k)} \end{bmatrix} \quad , \quad \Delta C^{(k)} = \begin{bmatrix} C_1 - (f_1)^{(k)} \\ C_2 - (f_2)^{(k)} \\ \vdots \\ C_n - (f_n)^{(k)} \end{bmatrix}$$

$$J^{(k)} = \begin{bmatrix} \left(\frac{\partial f_1}{\partial x_1}\right)^{(k)} & \left(\frac{\partial f_1}{\partial x_2}\right)^{(k)} & \cdots\cdots \left(\frac{\partial f_1}{\partial x_n}\right)^{(k)} \\ \left(\frac{\partial f_2}{\partial x_1}\right)^{(k)} & \left(\frac{\partial f_2}{\partial x_2}\right)^{(k)} & \cdots\cdots \left(\frac{\partial f_2}{\partial x_n}\right)^{(k)} \\ \vdots & \vdots & \vdots \\ \left(\frac{\partial f_n}{\partial x_1}\right)^{(k)} & \left(\frac{\partial f_n}{\partial x_2}\right)^{(k)} & \cdots\cdots \left(\frac{\partial f_n}{\partial x_n}\right)^{(k)} \end{bmatrix}$$

ماتریس $J^{(k)}$ ، ماتریس ژاکوبین نامیده می‌شود. عناصر این ماتریس مشتقات جزئی هستند که در $x^{(k)}$ ها ارزیابی می‌شوند. فرض می‌شود که $J^{(k)}$ در هر تکرار معکوس پذیر باشد. اعمال روش

نیوتن رافسون به دستگاه معادلات غیر خطی ، مسأله را به حل دستگاه معادلات خطی کاهش می‌دهد تا مقادیری که دقت تخمین ها را بهبود می‌دهند. تعیین شوند.

روشی برای وارون کردن ماتریس ژاکوبین :

روشی که برای وارون نمودن ماتریس ژاکوبین انتخاب شده است ، روشی است که در محاسبات عددی به آن پرداخته می‌شود. این روش به نام کیلی –ها میلتون موسوم است.

برای وارون نمودن ژاکوبین به روش کیلی‌هامیلتون ، ابتدا می‌بایست چند جمله‌ای مشخصه ماتریس بدست آید. برای محاسبه چند جمله‌ای مشخصه یک ماتریس نیز از روش لوری یر استفاده شده است. بنابراین در بدو امر می‌بایست ، تعیین چند جمله‌ای مشخصه یک ماتریس به روش لوری‌یر تشریح و سپس به وارون نمودن ماتریس پرداخت. روش لوری یر برای تعیین چند جمله‌آی مشخصه ماتریس ژاکوبین :

این روش از این مطلب استفاده می‌کند که اگر λ مقدار ویژه ماتریس A باشد λ^i مقدار ویره A^i است. هرگاه فرض کنیم λ_k مقدار ویژه A است و قرار دهیم :

$$S_i = \sum_{k=1}^{n} \lambda_k^i = \lambda_1^i + \lambda_2^i + ... + \lambda_n^i$$

در این صورت λ_k^i ها مقادیر ویژه A^i هستند. همچنین S_i قابل محاسبه است زیرا می‌توان نشان داد که :

$$S_i = tr\left(A^i\right)$$

که در آن S_i ها مجموع درایه‌های قطر اصلی A^i ها می‌باشد. بعلاوه هرگاه چند جمله‌ای مشخصه A بصورت زیر باشد :

$$f(\lambda) = \lambda^n + P_1\lambda^{n-1} + P_2\lambda^{n-2} + ... + P_{n+1}\lambda + P_n = ۰$$

رابطه بین S_i ها و P_k به قرار زیر است :

$$\begin{cases} P_1 = -S_1 \\ P_k = \dfrac{1}{K}\left(S_k + P_1 S_{k-1} + P_2 S_{K-2} + ... + P_{K-1} S_1\right) \\ \qquad K = 2,3,.....n \end{cases}$$

لذا با استفاده از رابطه بازگشتی فوق و داشتن P_1 می‌توان P_k ها را حساب کرد و از آن چند جمله‌ای مشخصه ماتریس A تعیین می‌شود.

تعیین وارون ژاکوبین با استفاده از روش کیلی – هامیلتون :

هر گاه چند جمله‌ای مشخصه ماتریس A بصورت زیر باشد :

$$f(\lambda) = \lambda^n + P_1\lambda^{n-1} + P_2\lambda^{n-2} + ... + P_{n-1}\lambda + P_n = 0$$

بنا قضیه کیلی – هامیلتون داریم :

$$A^n + P_1 A^{n-1} + P_2 A^{n-2} + ... + P_{n-1} A + P_n I = 0$$

هرگاه $|A| \neq 0$ باشد ، با ضرب A^{-1} در طرفین رابطه فوق خواهیم داشت :

$$A^{n-1} + P_1 A^{n-2} + P_2 A^{n-3} + ... + P_{n-1} I + P_n A^{-1} = 0$$

و از آنجا خواهیم داشت :

$$A^{-1} = -\frac{1}{P_n}\left[A^{n-1} + P_1 A^{n-2} + P_2 A^{n-3} + ... + P_{n-1} I\right]$$

بدین ترتیب ماتریس A وارون خواهد شد. که در آن هرگاه P_n برابر با صفر شود ، ماتریس فوق وارون پذیر نمی‌باشد.

فصل سوم

معادلات حل پخش بار به روش نیوتن – رافسون

حل پخش بار به روش نیوتن – رافسون

روش نیوتن – رافسون از لحاظ ریاضی ، بخاطر همگرایی درجه دوم آن ، نسبت به روش گـوس – سایدل برتری دارد و احتمال واگرایی آن در مسایل بدساختار کمتر است. در سیـستم‌هـای قـدرت بزرگ ، روش نیوتن – رافسون مؤثرتر و عملی‌تر می‌باشد. تعداد تکرارهای لازم برای تعیین پاسخ ، مستقل از اندازه سیستم بوده ، ولی ارزیابی‌های تابعی بیشتری در هـر تکـرار مـورد نیـاز اسـت. از آنجایی که در مسئله پخش بار ، توان حقیقی و اندازه ولتاژ ، در شین‌هـای بـا ولتـاژ کنتـرل شـده معلوم است ، معادله پخش بار ، بصورت قطبی فرمول بندی مـی‌شـود. بـرای یـک شـین نـوعی از سیستم قدرت ، جریان ورودی به شین i به وسیله رابطه زیر داده می‌شود :

$$I_i = \sum_{j=1}^{n} Y_{ij} V_j$$

در رابطه بالا ، j شامل شین i نیز می‌شود. با بیان این رابطه بصورت قطبی ، داریم :

$$I_i = \sum_{j=1}^{n} \left|Y_{ij}\right|\left|V_j\right| \quad \left(\theta_{ij} + S_i\right) \qquad (1)$$

توان مختلط در شین i برابر است با :

$$S_i = V_i \cdot_i^* \qquad (2)$$

یا

$$S_i^* = V_i^* I_i = P_i - jQ_i \qquad (3)$$

با جایگزینی I_i از رابطه (۱) در رابطه (۳) خواهیم دانست :

$$P_i - jQ_i = \left|V_i\right| < -S_i \sum_{j=1}^{n} \left|Y_{ij}\right|\left|V_j\right| < \left(\theta_{ij} + S_i\right) \qquad (4)$$

با جداسازی قسمتهای حقیقی و موهومی این معادله می‌توان نوشت :

$$P_i = \sum_{j=1}^{n} |V_i||V_j||Y_{ij}| \cos(\theta_{ij} - S_i + S_j) \qquad (5)$$

$$Q_i = -\sum_{j=1}^{n} |V_i||V_j||Y_{ij}| \sin(\theta_{ij} - S_i + S_j) \qquad (6)$$

روابط (۵) و (۶) را بصورت زیر مرتب می‌کنیم :

$$P_i = \sum_{j=1}^{n} |V_i||V_j||Y_{ij}| \cos(S_i - S_j - \theta_{ij}) \qquad (7)$$

$$Q_i = \sum_{j=1}^{n} |V_i||V_j||Y_{ij}| \sin(S_i - S_j - \theta_{ij}) \qquad (8)$$

معادلات (۷) و (۸) شامل مجموعه‌ای از معادلات جبری غیر خطی برحسب متغیرهای مستقل هستند ، که اندازه ولتاژها برحسب pu و زاویه‌های فاز برحسب رادیان می‌باشد. برای هر شین دو معادله وجود دارد. که با روابط (۷) و (۸) نشان داده شده است. برای هر شین با ولتاژ کنترل شده نیز یک معادله وجود دارد که با رابطه (۷) مشخص می‌شود. با بسط معادلات (۷) و (۸) بصورت سری تیلور در نزدیکی تخمین اولیه و با چشم پوشی از جملات با درجات بالاتر ، مجموعه معادلات خطی زیر بدست می‌آید :

$$\begin{bmatrix} \Delta P_2^{(k)} \\ \vdots \\ \Delta P_n^{(k)} \\ \Delta Q_2^{(k)} \\ \vdots \\ \Delta Q_n^{(k)} \end{bmatrix} = \left[\begin{array}{ccc|ccc} \frac{\partial P_2^{(k)}}{\partial S_2} & \cdots & \frac{\partial P_2^{(k)}}{\partial S_n} & \frac{\partial P_2^{(k)}}{\partial |V_2|} & \cdots & \frac{\partial P_2^{(k)}}{\partial |V_n|} \\ \vdots & & & & & \\ \frac{\partial P_n^{(k)}}{\partial S_2} & \cdots & \frac{\partial P_n^{(k)}}{\partial S_n} & \frac{\partial P_n^{(k)}}{\partial |V_2|} & \cdots & \frac{\partial P_n^{(k)}}{\partial |V_n|} \\ \hline \frac{\partial Q_2^{(k)}}{\partial S_2} & \cdots & \frac{\partial Q_2^{(k)}}{\partial S_n} & \frac{\partial Q_2^{(k)}}{\partial |V_2|} & \cdots & \frac{\partial Q_2^{(k)}}{\partial |V_n|} \\ \vdots & & & \vdots & & \\ \frac{\partial Q_n^{(k)}}{\partial S_2} & \cdots & \frac{\partial Q_n^{(k)}}{\partial S_n} & \frac{\partial Q_n^{(k)}}{\partial |V_2|} & \cdots & \frac{\partial Q_n^{(k)}}{\partial |V_n|} \end{array}\right] \begin{bmatrix} \Delta S_2^{(k)} \\ \vdots \\ \Delta S_n^{(k)} \\ \Delta |V_2|^{(k)} \\ \vdots \\ \Delta |V_n|^{(k)} \end{bmatrix}$$

در معادلات بالا ، شین ۱ بعنوان شین مرجع در نظر گرفته شده است. ماتریس ژاکوبین رابطه خطی شده تغییرات کوچک زاویه ولتاژ $\Delta S_i^{(k)}$ و اندازه ولتاژ $\Delta |V_i^{(k)}|$ را برحسب تغییرات کوچک توان‌های اکتیو و راکتیو $\Delta Q_i^{(k)}, \Delta P_i^{(k)}$ می‌دهد. عناصر ماتریس ژاکوبین مشتقات جزئی معادلات (۷) و (۸) هستند که به ازای $\Delta |V_i|^{(k)}, \Delta S_i^{(k)}$ ارزیابی می‌شوند رابطه اخیر را می‌توان به صورت خلاصه شده زیر نوشت :

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \begin{bmatrix} J_1 & J_2 \\ J_3 & J_4 \end{bmatrix} \begin{bmatrix} \Delta S \\ \Delta |V| \end{bmatrix} \qquad (9)$$

در شین‌های بار (PQ) و انتقال (transbus) ، اندازه ولتاژها مجهول می‌باشد. بنابراین اگر به تعدادm ، دارای شین‌های بار و انتقال باشیم ، در اینصورت. n معادله شامل $\Delta V, \Delta Q$ خواهد بود. ΔS ها برای تمامی شینها بجز شین مرجع مجهول است و اگر n شین در شبکه موجود باشد بنابراین (n-1) معادله شامل $\Delta S, \Delta P$ است بدین ترتیب تعداد قیود توان اکتیور (n-1) و تعداد قیود توان راکتیو (n-1-m) بوده و ماتریس ژاکوبین دارای ابعاد (n-1+m) (n-1+m) خواهد

بود. ماتریس J_1 دارای ابعاد $(n-1)(n-1)$ $J_2,(n-1)(m)$ دارای $J_3(n-1)(m)$ دارای $J_4,(m)(n-1)$ و (m) (m) می‌باشد.

عناصر قطری و غیر قطری J_1 عبارتند از :

$$\frac{\partial P_i}{\partial S_i} = \sum_{j\neq i} |V_i||V_j||Y_{ij}|\sin(\theta_{ij} - S_i + S_j) \qquad (10)$$

$$\frac{\partial P_i}{\partial S_j} = -|V_i||V_j||Y_{ij}|\sin(\theta_{ij} - S_i + S_i) j \neq i \qquad (11)$$

عناصر قطری و غیر قطری J_2 عبارتند از :

$$\frac{\partial P_i}{\partial |V_i|} = 2|V_i||Y_{ii}|\cos\theta_{ii} + \sum_{j\neq i} |V_j||Y_{ij}|\cos(\theta_{ij} - S_i + S_j) \qquad (12)$$

$$\frac{\partial P_i}{\partial |V_j|} = |V_i||Y_{ij}|\cos(\theta_{ij} - S_i + S_j) j \neq i \qquad (13)$$

عناصر قطری و غیر قطری J_3 عبارتند از :

$$\frac{\partial Q_i}{\partial S_i} = \sum_{j\neq i} |V_i||V_j||Y_{ij}|\cos(\theta_{ij} - S_i + S_j) \qquad (14)$$

$$\frac{\partial Q_i}{\partial S_i} = -|V_i||V_j||Y_{ij}|\cos(\theta_{ij} - S_i + S_j) j \neq i \qquad (15)$$

و عناصر قطری و غیر قطری J_4 نیز عبارتند از :

$$\frac{\partial Q_i}{\partial |V_i|} = -2|V_i||Y_{ii}|\sin\theta_{ii} - \sum_{j\neq i} |V_j||Y_{ij}|\sin(\theta_{ij} - S_i + S_j) \qquad (16)$$

$$\frac{\partial Q_i}{\partial |V_j|} = -|V_i||Y_{ij}|\sin(\theta_{ij} - S_i + S_j) j \neq i \qquad (17)$$

جملات $\Delta Q_i^{(k)}, \Delta P_i^{(k)}$ اختلاف بین مقادیر برنامه ریزی شده و محاسبه شده هستند که به تـوان باقیمانده موسوم بوده و با روابط زیر نشان داده می‌شوند :

$$\Delta P_i^{(k)} = P_i^{sch} - P_i^{(k)} \qquad (18)$$

$$\Delta Q_i^{(k)} = Q_i^{sch} - Q_i^{(k)} \qquad (19)$$

تخمین های جدید ولتاژها عبارتند از :

$$S_i^{(k+1)} = S_i^{(k)} + \Delta S_i^{(k)} \qquad (20)$$

$$\left|V_i^{(k+1)}\right| = \left|V_i^{(k)}\right| + \Delta\left|V_i^{(k)}\right| \qquad (21)$$

بنابراین مراحل حل پخش بار با استفاده از الگوریتم نیوتن – رافسون مطابق زیر است :

۱) برای شین‌های بار که Q_i^{sch}, P_i^{sch} معلوم هستند ، اندازه ولتاژها و زاویه‌های فـاز آنهـا مـساوی مقادیر شین مرجع یـا ۱ و ۰ در نظـر گرفتـه مـی‌شـوند. یعنـی $\left|V_i\right|^{(o)} = 1$ و $S_i^{(o)} = ۰$ در شـین نیروگاهی $P_i^{sch}, \left|V_i\right|$ معلوم هستند ، زاویه‌های فاز انها را مساوی زاویه شین مرجـع یـا 0 در نظـر می‌گیریم. $S_i^{(o)} = ۰$

۲) برای شین‌های بار $Q_i^{(k)}, P_i^{(k)}$ به ترتیب از معادلات (۷) و (۸) محاسبه شده و $\Delta Q_i^{(k)}, \Delta P_i^{(k)}$ به ترتیب از روابط (۱۸) و (۱۹) بدست می‌آیند.

۳) برای شین‌های با ولتاژ کنتـرل شـده ، $\left|V_i\right|$ معلـوم و $S_i^{(o)} = ۰$ در نظـر مـی‌گیـریم و $P_i^{(k)}$ و $\Delta P_i^{(k)}$ به ترتیب از معادلات (۷) و (۱۸) بدست می‌آیند.

۴) عناصر ماتریس ژاکوبین $\left(J_4, J_3, J_2, J_1\right)$ از معادلات (۱۰) تا (۱۷) محاسبه می‌شوند.

۵) ماتریس ژاکوبین وارون و سپس تقریب‌های جدید بدست می آیند.

۶) اندازه‌ها و زاویای فاز جدید ولتاژ با استفاده از معادلات (۲۰) و (۲۱) محاسبه می‌شوند.

۷) این فرآیند آنقدر ادامه می‌یابد تا پس ماندهای $\Delta Q_i^{(k)}, \Delta P_i^{(k)}$ کوچکتر از دقت تعیین شده گردند.

$$\left|\Delta P_i^{(k)}\right| \le \varepsilon$$
$$\left|\Delta Q_i^{(k)}\right| \le \varepsilon$$

فصل چهارم

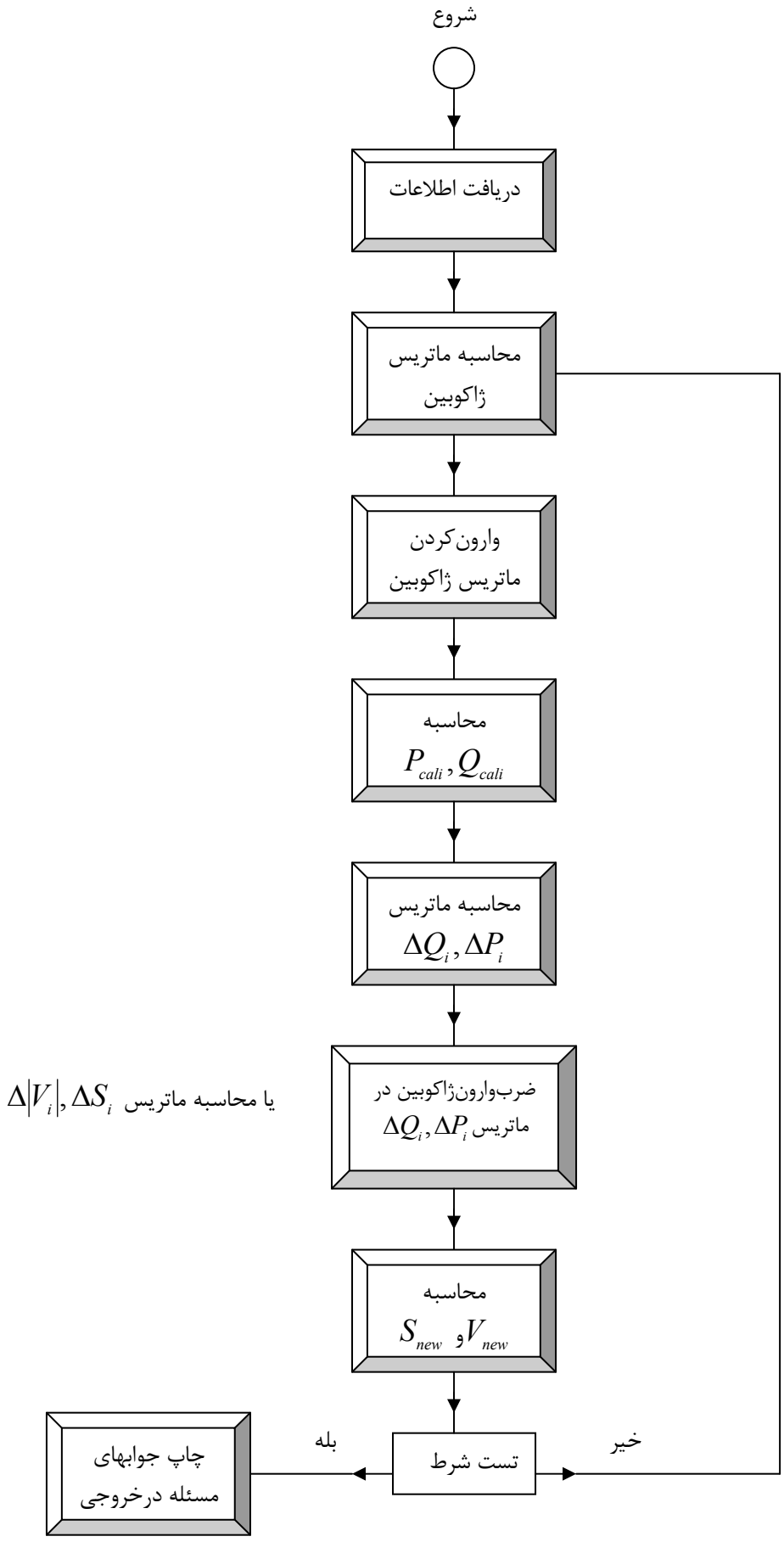
تعیین الگوریتم کلی برنامه

الگوریتم کلی برنامه :

برای حل یک مسئله در هر بنامه نویسی ابتدا به ساکن نیاز به ایجاد یک الگوریتم خاص برای تعیین روش انجام مراحل برنامه می‌باشد. بر همین مبنا می‌بایست مسئله را به اجزاء کوچکتر تقسیم نمود و سپس این اجزاء کوچکتر را حل نمود. قسمتهای کوچکتر ممکن است خود شامل مراحل مختلف باشد که برای ساده‌تر شدن بهتر است این قسمتها کوچکتر شده نیز به چند قسمت دیگر تفکیک شود. هر جزء در کنار جزء دیگر می‌تواند بخشی از راه حل را در اختیار قرار دهد و بخشهای مختلف در ساختار کلی الگوریتم پاسخی برای حل مسئله می‌باشد.

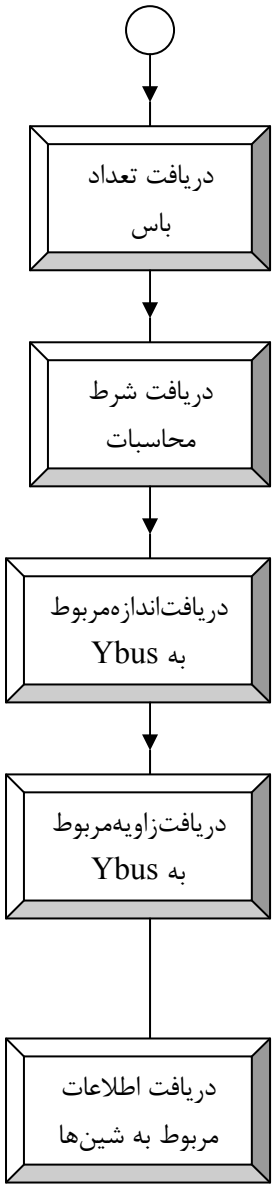
الگوریتم کلی در اینجا حل معادلات جبری غیر خطی به روش نیوتن – رافسون می‌باشد که در روشهای محاسبات عددی به آن پرداخته می‌شود. و جواب نهایی در تکرارهای متوالی حاصل می‌شود. همانطور که در قبل عنوان شد. اولین گام ایجاد ماتریس ژاکوبین می‌باشد. که این ماتریس دارای درایه‌هایی با مشتقات جزئی است در مرحله بعد ماتریس ژاکوبین می‌بایست وارون شود. و وارون ماتریس ژاکوبین در ماتریس حاصل از $\Delta Q_i, \Delta P_i$ ضرب شود تا جوابی برای تکرارها در محل مختلف برای مجهولات $\Delta|V_i|, \Delta S_i$ حاصل شود. بعد از محاسله $\Delta|V_i|, \Delta S_i$ ها شرط مسئله تست شده ، اگر از مقدار تعیین شده کوچکتر باشد روند تکرار متوقف می‌شود در غیر اینصورت تمام مراحل از اول تکرار می‌شود. اگر شرط برقرار بود میزان اختلاف جواب مسئله با مقادیر اولیه جمع می‌شود و جواب نهایی برای $\Delta|V_i|, \Delta S_i$ بدست می‌آید بنابراین ابتدا به ساکن برنامه باید بتوان اطلاعات مورد نیاز خود را از ورودی دریافت دارد و سپس ماتریس ژاکوبین را تشکیل کند. در مرحله بعد ماتریس ژاکوبین وارون می‌شود. مرحله بعدی را از ضرب وارون ژاکوبین در ماتریس

$\Delta Q_i, \Delta P_i$ بدست می‌آید که جوابی برای $\Delta|V_i|, \Delta S_i$ می‌باشد. در نهایت شرط تست شده اگر این شرط ارضاء شده باشد تکرار را متوقف در غیر اینصورت فرآیند تکرار از سرگرفته می‌شود. زمانی که شرط برقرار شد نوبت به مرحله آخر که همان محاسبه توانهای اکتیو و راکتیو محاسباتی و تولیدی در شینها می‌باشد می‌رسد و در آخر کلیه جوابهای بدست آمده را باید در خروجی برنامه نمایش داد. بنابراین این الگوریتم کلی را در زیر خواهیم دید. که البته هر یک از این بخشها شامل اجزاء کوچکتر می‌باشد که به آنها اشاره می‌شود الگوریتم کلی برنامه در شکل زیر دیده می‌شود.


شروع
دریافت اطلاعات
محاسبه ماتریس ژاکوبین
وارون کردن ماتریس ژاکوبین
محاسبه P_{cali}, Q_{cali}
محاسبه ماتریس $\Delta Q_i, \Delta P_i$
ضرب وارون ژاکوبین در ماتریس $\Delta Q_i, \Delta P_i$
یا محاسبه ماتریس $\Delta|V_i|, \Delta S_i$
محاسبه V_{new} و S_{new}
تست شرط
بله
خیر
چاپ جوابهای مسئله درخروجی

الگوریتم دریافت اطلاعات در ورودی :

برنامه برای اینکه بتواند محاسبات را آغاز نماید نیاز به داشتن یک سری اطلاعات اولیه دارد. این اطلاعات شامل مقادیر اندازه ادمیتانس باس (Ybus) زاویه ادمیتانس باس و اطلاعات مربوط به شینهای شبکه مورد مطالعه می‌باشد. ادمیتانس باس (Ybus) اطلاعات مربوط به خطوط انتقال نیروی شبکه مورد مطالعه را در اختیار نرم افزار قرار می‌دهد. بنابراین اگر بخواهیم بخش مربوط به دریافت اطلاعات در ورودی را به قسمتهای کوچکتر تفکیک نماییم. باید گفت اولین چیزی که باید در اختیار نرم افزار قرار گیرد تعداد باس بارها بوده تا نرم افزار بداند به چه اندازه‌ای باید ماترس Ybus و دیگر ماتریسهای مورد استفاده در آن را تشکیل دهد. بعد از تعداد باس شرط مسئله را نیز می‌توان در یک متغیر قرار داد تا در انتهای برنامه تست شود. قسمت بعدی مربوط به دریافت اندازه درایدهای Ybus شبکه می‌باشد. در مرحله بعد زاویه مقادیر Ybus وارد خواهد شد و سپس نوبت به دریافت اطلاعات شینهای شبکه می‌رسد ت اطلاعات مورد نیاز نرم افزار کامل شود. در زیر فلوچارت الگوریتم دریافت اطلاعات مشاهده می‌شود.

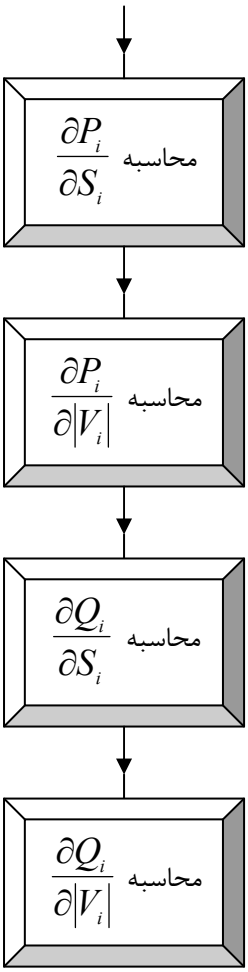


فلوچارت مربوط به دریافت اطلاعات

الگوریتم محاسبه ماتریس ژاکوبین :

درایه‌های ماترس ژاکوبین هر کدام مشتقات جزئی می‌باشد. این مشتقات جزئی شامل $\frac{\partial P_i}{\partial S_i}, \frac{\partial P_i}{\partial |V_i|}$ و همچنین شامل $\frac{\partial Q_i}{\partial S_i}, \frac{\partial Q_i}{\partial |V_i|}$ می‌باشد. بنابراین ماتریس ژاکوبین به چهار قسمت تقسیم می‌شود. که هر یک مطابق الگوریتم خاص خود باید محاسبه شود. از آنجا که S_i ها برای

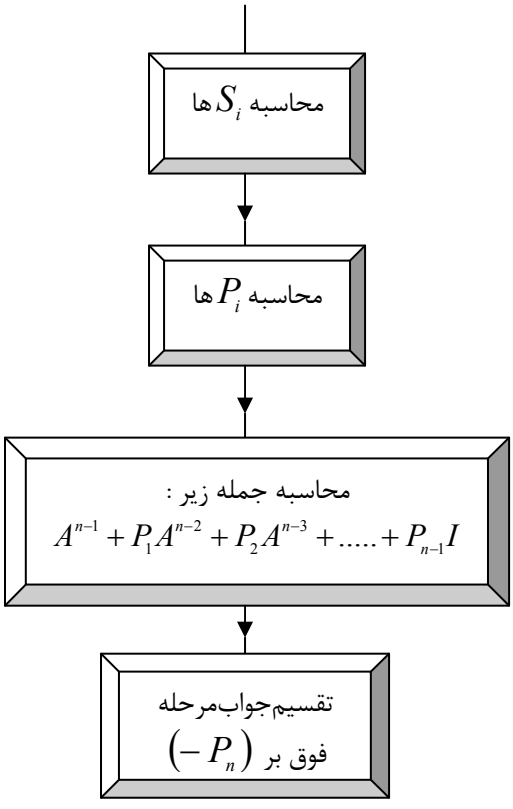
تمامی شین‌ها به جزء شین اسلک مجهول است می‌بایست $\frac{\partial P_i}{\partial S_i}$ برای تمامی شین‌ها به جزء اسلک محاسبه شود. ولی $\frac{\partial Q_i}{\partial |V_i|}, \frac{\partial Q_i}{\partial S_i}, \frac{\partial P_i}{\partial |V_i|}$ فقط برای شین‌هایی محاسبه می‌شود که دارای ولتاژهای مجهول می‌باشند. که این شینه‌ها ، شین‌های trans bus , load bus می‌باشند. بنابراین قسمتهای مربوط به محاسبه ماتریس ژاکوبین در شکل زیر نمایش داده شده است.



الگوریتم مربوط به محاسبه وارون ژاکوبین :

این قسمت از برنامه که یکی از مشکلترین مراحل برنامه برنامه‌نویسی این نرم افـزار مـی‌باشـد و از روشی پی روی می‌کند که در محاسبات عددی به روش کیلی – هامیلتون شهرت دارد. در مراحـل اولیه انجام پروژه از روش دیگری استفاده شده بود که پاسخگوی حجم زیاد محاسبات و تکرارهـای مکرر نبود. بنابراین ناگزیر به متوسل شدن به روشی قدرتمندتر احساس شد و پـس از جـستجو در کتب مختلف محاسبات عددی این روش را بعنوان وارون کننـده مـاتریس ژاکـوبین انتخـاب شـد. مراحل ریاضی این روش در قسمتهای قبلی عنوان شده است.

قسمتهای مختلف وارون کردن ماتریس ژاکوبین شامل محاسله S_i ها و سـپس P_i هـا و توانهـای مختلف ماتریس ژاکوبین تا عددی برابر با یک واحد کوچکتر از ابعاد خود ماتریس و همچنین ضرب نمودن P_i ها در توانهایی از ماتریس ژاکوبین و در نهایت جمع همگی این جملات و تقسیم جواب بدست آمده بر $(-P_n)$ الگوریتم کلی این بخش از برنامه در شکل زیر مشاهده می‌شود.



فلوچارت الگوریتم برای وارون نمودن ژاکوبین

البته در فرآیند مراحل نرم افزار عمل وارون کردن ماتریس ژاکوبین به عهده یک تابع قـرار گرفتـه است که این تابع رابه نامه Inverse-matrix نام گذاری کرده‌ام. بنابراین هرگاه کنترل برنامه بـه این تابع رسید ، کدهای نوشته شده در آنرا اجرا خواهد کرد تا ژاکوبین را وارون نماید.

الگوریتم مربوط به محاسبه Q_{cali}, P_{cali} :

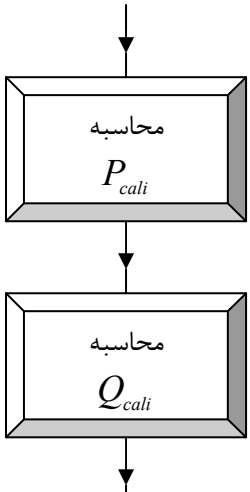
این قسمت از برنامه خود از دو بخش مجزا تشکیل شده است که یکی بـرای محاسـبه P_{cali} هـای تمامی شین‌ها به جزء شین اسلک و دیگری محاسبه Q_{cali} ها بـرای شـین‌هـایی کـه دارای ولتـاژ مجهول می‌باشند. برای این بخش دو تابع اختصاص یافته است. تـابع اول بنـام $Full - P_{cal}$، کـه محاسبات مربوط به P_{cali} ها را بعهده دارد و تابع دوم بنام $Full - Q_{cal}$ ، که محاسبات مربوط به Q_{cali}را انجام می‌دهد.

فرمولهای ریاضی آنها بصورت زیر است :

$$P_{cali} = \sum_{k=1}^{n} |V_i|.|V_k|.Y_{ik}.\cos(S_i - S_k - \theta_{ik})$$

$$Q_{cali} = \sum_{k=1}^{n} |V_i|.|V_k|.Y_{ik}.Sin(S_i - S_k - \theta_{ik})$$

باید توجه داشت که Q_{cali}را فقط برای شینهایی بدست می‌آوریم که این شین‌ها ولتاژهای مجهول دارند. بنابراین نرم افزار باید قابلیت تشخیص در خصوص یافتن شین‌هایی که ولتاژ مجهول هستند را داشته باشد. لازم به ذکر است که این شین‌ها ، شـینهای trans bus , load bus مـی‌باشـند. فلوچارت مربوط به الگوریتم این بخش از برنامه در شکل زیر دیده می‌شود.



فلوچارت مربوط به محاسبه Q_{cali}, P_{cali}

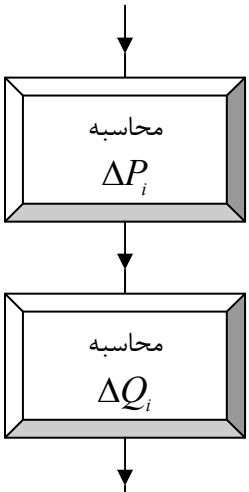
الگوریتم مربوط به محاسبه ماتریس $\Delta Q_i, \Delta P_i$:

این قسمت از برنامه توسط تابعه‌ای بنام full-del-P-Q انجام می‌پذیرد. این تابع آرایه‌ای بنام del-P-Q را پر می‌کند که بخشی از درایه‌های آنرا ΔP_i ها بخشی دیگر از درایه‌ها را ΔQ_i ها تشکیل می‌دهند. فرمول مورد استفاده برای محاسبه Δ_{Pi} ها و ΔQ_i ها بصورت زیر است :

$$\Delta P_i = P_{Gi} - P_{Li} - P_{cali}$$

$$\Delta Q_i = Q_{Gi} - Q_{Li} - Q_{cali}$$

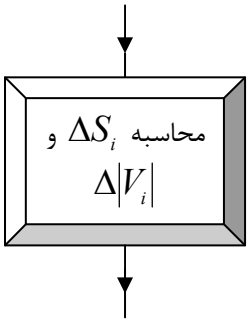
باید توجه داشت که ΔQ_i ها فقط برای شینهایی محاسبه می‌شوند که ولتاژ این شین‌ها جزء مجهولات مسئله است. فلوچارت الگوریتم این قسمت در شکل زیر نمایان است.



فلوچارت الگوریتم برای $\Delta Q_i, \Delta P_i$

الگوریتم مربوط به ضرب وارون ژاکوبین در ماتریس $\Delta Q_i, \Delta P_i$: (محاسبه $\Delta|V_i|, \Delta S_i$)

هرگاه ماتریس وارون ژاکوبین را در ماتریس $\Delta Q_i, \Delta P_i$ ضرب نماییم جوابهایی برای ΔS_i ها و $\Delta|V_i|$ بدست خواهیم آورد. در فرآیند برنامه نویسی ، این بخش توسط تابعه‌ای تحت عنوان full-del-delta-V انجام می‌پذیرد که فلوچارت الگوریتم آن همان فلوچارت نشان داده شده در الگوریتم اصلی برنامه می‌باشد.

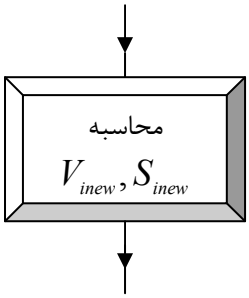


الگوریتم مربوط به محاسبه $|V_i|_{new}, S_{new_i}$:

انجام این مرحله بسیار ساده بوده و فقط کافیست ΔS_i جدید را با مقدار زاویه‌های محاسبه شده از تکرار قبلی جمع نمود تا مقدار $S_{i\ new}$ بدست آید. و برای $|V_i|_{new}$ نیز می‌بایست $\Delta|V_i|$ جدید با V_i های بدست‌آمده از تکرار قبلی جمع شود تا در نهایت مقدار $|V_i|_{new}$ بدست‌آید. فرمولهای مربوط به قرار زیرند :

$$S_{inew} = \Delta S_{inew} + S_{iold}$$
$$|V_i|_{new} = \Delta|V_i|_{new} + |V_i|_{old}$$

فلوچارت الگوریتم همان فلوچارت مربوط به الگوریتم اصلی برنامه است. که در زیر دیده می‌شود.



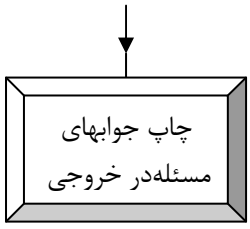
فلوچارت مربوط به محاسبه $|V_i|_{new}, S_{i\,new}$

الگوریتم تست شرط :

این مرحله نیز ساده بوده و فقط با تست یک شرط انجام می‌پذیرد. اگر شرط ارضاء نشد تکرار ادامه می‌یابد ولی اگر شرط ارضاء شده باشد تکرارها خاتمه یافته و به اصطلاح سیستم converge می‌کند.

الگوریتم مربوط به چاپ جوابهای مسئله در خروجی :

بعد از converge نمودن سیستم دیگر مجهولات سیستم که شامل PG_i ها و QG_i می باشد محاسبه شد و در نهایت تمامی مقادیر محاسبه شده توسط الگوی خروجی در صفحه نمایش مانیتور چاپ می‌شود. نمایش فلوچارت آن در شکل زیر نشان داده شده است.



فلوچارت مربوط به چاپ خروجی

فصل پنجم

مروری بر دستورات برنامه نویسی C++

دستورات C++ بکار رفته در نرم افزار :

انواع داده

هدف از برنامه نویسی ، ورود داده‌ها به کامپیوتر ، پردازش داده‌ها و استخراج نتایج است. لذا داده‌ها نقش مهمی را در برنامه نویسی ایفا می‌کنند. یکی از جنبه‌های زبانهای برنامه سازی که باید دقیقاً مورد بررسی قرار گیرد. انواع داده‌هایی است که آن زبان با آنها سروکار دارد. در زبان C++ پنج نوع داده وجود دارد که عبارتند از : char , int , float , double نوع char برای ذخیره داده‌ه ای کاراکتری مثل a , b , x بکار می‌رود. نوع int برای ذخیره اعداد صحیح مثل ، , 125 , 1250 10 بکار می‌رود. نوع float برای ذخیره اعداد اعشاری مثل 12.5 , 16.25 , 1250.12 بکار می‌رود. و نوع double برای ذخیره اعداد اعشاری که بزرگتر از نوع float باشند مورد استفاده قرار می‌گیرد. هر یک از انواع داده‌های char , int , float , double مقایدری را می‌پذیرند که ممکن است از پردازنده‌ای (CPU) به پردازنده دیگر متفاوت باشد. بعنوان مثال طول نوع int در محیطهای 16 بینی مثل Dos یا ویندوز 3.1 دو بایت و در محیطهای 32 بیتی مثل ویندوز NT ، چهار بایت است. بنابراین اگر برنامه قرار است در محیطهای مختلفی استفاده شود بهتر است از کوچکترین مقدار انواع استفاده شود.

متغیرها

متغیر نامی برای کلمات حافظه است که داده‌ها در آن قرار می‌گیرند و ممکن است مقدار آن در طول اجرای برنامه تغییر کند. برای مراجعه به متغیرها از نام آنها استفاده می‌شود. لذا متغیرها امکان نامگذاری برای کلمات حافظه (بایتها) را فراهم می‌کنند. برای نامگذاری متغیرها می‌توان از

ترکیبی از حروف a تا z یا A تا Z ، ارقام و خط ربط (-) استفاده کرد ، به شرط آنکه اولین کاراکتر آنها رقم نباشد. نام متغیر می‌تواند با هر طولی باشد ولی فقط 31 کارکتر اول آن مورد استفاده قرار میگیرد.

تعریف متغیر

همانطور که عنوان شد ، متغیرها محل ذخیره داده‌ها هستند و چون داده‌ها دارای نوع‌اند ، متغیرها نیز باید دارای نوع باشند. بعبارت دیگر متغیرهای فاقد نوع در C++ شناخته نیستند. قبل از بکار گرفتن متغیرها باید نوع آنها را مشخص نمود. نوع متغیر مقداری را که متغیر می‌تواند بپذیرد و اعمالی را که می‌توانند بر روی آن مقادیر انجام شوند ، مشخص می‌کند. تعیین نوع متغیر را تعریف متغیر گویند. برای تعیین نوع متغیر ، بصورت زیر عمل می‌شود :

نوع داده نام متغیر :

مقدار دادن به متغیرها :

برای مقدار دادن به متغیرها به سه روش می‌توان عمل کرد :

۱) هنگام تعریف (تعیین نوع) متغیر

مثال : int x,y = 5;

Char ch = a ;

۲) پس از تعریف نوع متغیر و با دستور انتخاب (=)

مثال : int x ;

Float y ;

X = 5;

Y=2.5;

۳) در دستورات ورودی.

مثال : int x,y;

Scant ("%d %d",& x , &y);

عملگرها

عملگرها نمادهایی هستند که اعمال خاصی را انجام می‌دهند. بعنوان مثال + عملگری است که دو مقدار را با هم جمع می‌کند. پس تعریف متغیرها و مقدار دادن به آنها باید بتوان عملیاتی را روی آنها انجام داد. برای انجام این عملیات باید از عملگرها استفاده کرد. عملگرها در زبان C++ به چند دسته تقسیم می‌شوند : ۱- عملگرهای محاسباتی ۲- عملگرهای رابطه‌ای ۳- عملگرهای منطقی ۴- عملگرهای بیتی. عملگرها بر روی یک یا دو مقدار عمل می‌کنند. مقادیری که عملگرها بر روی آنها عمل می‌کنند ، عملوند نام دارند بعنوان مثال ، در a+5 متغیر a و عدد5 را عملوندهای عملگر + گویند.

عملگرهای محاسباتی

عملگرهای محاسباتی ، عملگرهایی هستند که اعمال محاسباتی را روی عملوندها انجام می‌دهند. هر یک از عملگرهای - ، + ، × ، / تقریباً در همه زبانها وجود دارد. عملگر ٪ برای محاسبه باقیمانده بکار می‌رود. این عملگر ، عملوند اول را بر عملوند دوم تقسیم می‌کند و باقیمانده را بر می‌گرداند. عملگر کاهش (--) یک واحد از عملوندش کم می‌کند و نتیجه را در آن عملوند قرار می‌دهد و عملگر افزایش (++) یک واحد به عملوندش اضافه کرده و نتیجه را در آن عملوند قرار می‌دهد. جدول زیر عملگرهای محاسباتی و خاصیت آنها را نمایش می‌دهد.

عملگر	نام	مثال
-	تفریق یکانی	x- یا x-y
+	جمع	X+y
*	ضرب	y×X
/	تقسیم	x/y
٪	باقیمانده تقسیم	X%y
--	کاهش	--x یاx—
++	افزایش	++xیاX++

عملگرهای رابطه‌ای

عملگرهای رابطه‌ای ، ارتباط بین عملوندها را مشخص می‌کنند. اعمالی مثل تساوی دو مقدار ، کوچکتری یا بزرگتر بودن ، مقایسه با صفر ، و غیره ، توسط عملگرهای رابطه‌ای مشخص می‌شود. عملگرهای رابطه‌ای درجدول زیر آمده‌اند. عملگر == در دستورات شرطی برای مقایسه و مقدار مورد استفاده قرار می‌گیرد. بعنوان مثال ، دستور مقایسه دو مقدار y,x باید بصورت آیا x(x= =y) مساوی y است نوشته شود.

عملگر	نام	مثال
>	بزرگتر	x>y
>=	بزرگتر یا مساوی	x>=y
<	کوچکتر	X<y
<=	کوچکتر یا مساوی	X<=y
= =	متساوی	x= =y
= !	نامساوی	X!=y

عملگرهای منطقی

عملگرهای منطقی بر روی عبارات منطقی عمل می‌کنند. عبارات منطقی دارای دو ارزش درستی و نادرستی‌اند. ارزش نادرستی با مقدار صفر و ارزش درستی با مقدار یک مشخص می‌شود. عملگرهای منطقی در جدول شکل زیر آمده‌اند. نتیجه عملگر ! وقتی درست است که عملوند آن دارای ارزش نادرستی باشد. نتیجه عملگر && وقتی درست است که هر دو عملوند ارزش درستی داشته باشند. و نتیجه عملگر || وقتی نادرست است که هر دو عملوند ارزش نادرستی داشته باشند و در بقیه موارد ارزش درستی دارد.

عملگر	نام	مثال
!	نقیض (not)	!x
&&	و (and)	x>y&&m<p
||	یا (or)	X<y||m<p

عملگر siceof

این عملگر ، یک عملگر زمان ترجمه است و می‌تواند طول یک متغیر یا نوع داده را بر حسب بایت تعیین کند. بعنوان مثال اگر ندانیم نوع int در کامپیوتری خاص چند بایت است ، با این عملگر می‌توان به آن پی برد. این عملگر بصورتهای زیر بکار می‌رود.

sizeof متغیر ;

sizeof (نوع داده) ;

از این عملگر در دستورات تخصیص حافظه مثل malloc که از قبل مشخص نیست به چه میزان از حافظه را اشغال می‌کند استفاده می‌شود.

ساختار تکرار for

ساختار تکرار for یکی از امکانات ایجاد حلقه است و معمولاً در حالتی که تعداد دفعات تکرار حلقه از قبل مشخص باشد ، بکار می‌رود. در این ساختار متغیری وجود دارد که تعداد دفعات تکرار را کنترل می‌کند. این متغیر را شمارنده یا اندیس حلقه تکرار می‌گویند. اندیس حلقه دارای یک مقدار اولیه است. و در هر بار اجرای دستورات حلقه ، مقداری به آن اضافه می‌شود. این مقدار را که پس از هر بار اجرای حلقه به شمارنده اضافه می‌شود ، گام حرکت گویند ، گام حرکت می‌تواند عددی صحیح یا اعشاری ، مثبت یا منفی و یا کاراکتری باشد. یکی دیگر از اجزائی حلقه for ، شرط حلقه است. شرط حلقه مشخص می‌کند که دستورات داخل حلقه تاکی باید اجرا شوند اگر این شرط دارای ارزش درستی باشد ، دستورات داخل حلقه اجرا می‌شوند و گرنه کنترل برنامه از حلقه تکرار خارج می‌شوند. دستورات for را به دو شکل می‌توان بکار برد :

روش اول :

```
for ( مقدار اولیه اندیس حلقه ؛ شرط حلقه ؛ گام حلقه )
{
دستور ۱
دستور ۲
  ⋮
دستور n
}
```

روش دوم :

```
for (;;)
{
دستور ۱
دستور ۲
  ⋮
دستور n
}
```

```
}
```

روش دوم برای تکرار حلقه‌های بینهایت مورد استفاده قرار می‌گیرد (حلقه تکراری که پایان ندارد) برای خاتمه دادن به اجرای برنامه باید کلید CTRL+BREAK را از صفحه کلید فشار داد. اگر قرار باشد حلقه تکرار فقط یک دستور را اجرا نماید نیازی به قرار دادن {} نمی‌باشد.

ساختار تکرار while

ساختار تکرار while یکی دیگر از امکاناتی است که برای اجرای دستورات بکار می‌رود. این ساختار بصورت زیر قابل استفاده است :

```
while ( شرط)
{
دستور ۱
دستور ۲
      ⋮
دستور n
}
```

اگر شرط حلقه برقرار باشد ، دستورات داخل آن اجرا می‌شود. در غیر اینصورت کنترل برنامه از حلقه خارج می‌شود. اگر حلقه فقط یک دستور را اجرا می‌کند نیازی به قرار دادن {} نیست.

ساختار تکرار do ….. while

ساختار تکرار do…..while مانند ساختار while است. با این تفاوت که در ساختار while ، شرط در ابتدای حلقه تست می‌شود. در حالی که در do…..while شرط حلقه در انتهای حلقه تست می‌شود. بنابراین ، دستورات موجود در حلقه do…..while در هر حال ، حداقل یک بار اجرا می‌شوند. این ساختار بصورت زیر بکار می‌رود.

```
Dϕ
دستور ۱
دستور ۲
    ⋮
دستور n
}while ( شرط ) ;
```

اگر فقط یک دستور در داخل حلقه قرار داشته باشد نیازی به {} نیست.

ساختار تصمیم if

ساختار if که نام دیگرش ، دستور انتقال کنترل شرطی است ، شرطی را تست می‌کند و در صورتیکه آن شرط دارای ارزش درستی باشد. مجموعه‌ای از دستورات را اجرا می‌کند. این دستور بصورت زیر بکار می‌رود :

```
if ( شرط )
{  دستور ۱
    ⋮
دستور n
}
else
{
دستور ۱
    ⋮
دستور n
}
```

چنانچه شرط مورد بررسی درست باشد ، دستور یا دستورات بعد از if و گرنه دستور یا دستورات بعد از else اجرا می‌شوند. اگر بیش از یک دستور بعد از if و else بیاید ، آن دستورات باید در بین } و { قرار گیرند. دستورات if می‌تواند فاقد قسمت else باشند. در اینصورت چنانچه شرط

مورد بررسی ، درست باشد ، دستورات بعد از if اجرا می‌شوند و گرنه بدون اجرای این دستورات کنترل برنامه از if خارج می‌شود.

تابع ( ) printf

تابع ( ) printf که در فایل stdio.h قرار دارد ، برای چاپ اطلاعات در صفحه نمایش بکار می‌رود. اگر این تابع با موفقیت اجرا شود ، تعداد و کاراکترهایی را که به خروجی منتقل شده‌اند بر می‌گرداند و در صورت بروز خطا ، یک عدد منفی را بر می‌گرداند ، نحوه کاربرد این تابع بصورت زیر است :

printf ("<عبارت 1>" و <عبارت 2>)

در این تابع < عبارت 2> اطلاعاتی است که باید به خروجی منتقل شوند و < عبارت 1> می‌تواند شامل موارد زیر باشند :

۱)اطلاعاتی که باید عیناً در خروجی چاپ شوند.

۲) کاراکترهای تعیین کننده فرمت خروجی ، این کاراکترها نوع اطلاعاتی را که در < عبارت 2> ذکر شده‌اند و باید به خروجی بروند ، مشخص می‌کنند. کاراکترهای فرمت با علامت ٪ شروع می‌شوند. بعنوان مثال ، %f برای چاپ اعداد اعشاری بکار می‌روند. کاراکترهای فرمت در جدول زیر نمایش داده شده‌اند.

۳) کارکترهای کنترلی ، این کاراکترها شکل خروجی اطلاعات را مشخص می‌کنند. اینکه آیا تمام اطلاعات در یک سطر باشند یا در چند سطر چاپ شوند. آیا اطلاعات با فاصله خاصی از یکدیگر چاپ شوند یا خیر ، و مواردی از این قبیل ، توسط کاراکترهای کنترلی مشخص می‌شوند.

کاراکترهای کنترلی با \شروع می‌شوند بعنوان مثال n \ موجب می‌شود تا سـطرجاری ، رد شـود و چاپ اطلاعات از سطر جدیدی آغاز گردد. کاراکترهای کنترلی در جدول زیر نمایش داده شده‌انـد. در C++ تابع دیگری بنام cout وجود دارد که مشابه Printf عمل می‌کند و برای چاپ اطلاعات در خروجی استفاده می‌شود.

کاراکتر	نوع اطلاعات که باید چاپ شود.
%C	یک کاراکتر
% d	اعداد صحیح دهدهی مثبت و منفی
% f	اعداد اعشاری ممیز شناور
% S	رشته‌ای از کاراکترها
% e	نمایش علمی عدد همراه حرف e

جدول کاراکترهای فرمت.

کاراکتر	عملی که باید انجام شود.
\ f	موجب انتقال کنترل به صفحه جدید می‌شود.
\ n	موجب انتقال کنترل به خط جدید می‌شود.
\ t	انتقال به 8 محل بعدی صفحه نمایش
\ "	چاپ کوتیشن (")
\ v	انتقال کنترل به 8 سطر بعدی
\ o	Null (رشته تهلی)

کاراکترهای کنترلی

تابع Scanf ( )

این تابع برای ورودی اطلاعات از صفحه کلید مورد استفاده قرار می‌گیرد و یک تابع همه منظوره در ورود داده‌هاست. الگوی این تابع در فایل Stdio.h قرار دارد. این تابع تمام انواع داده‌ها را می‌تواند از ورودی بخواند و آنها را در حافظه ذخیره نماید. اگر این تابع با موفقیت اجرا شود ، تعداد متغیرهایی را که از ورودی خوانده است بر می‌گرداند ودر صورت بروز خطا ، EOF توسط تابع برگردانده می شود. EOF مقداری است که بیانگر عدم اجرای صحیح تابع ( ) scanf است. تابع Scanf بصورت زیر بکار می‌رود.

scanf (" <عبارت 1> و <عبارت 2>)

<عبارت 2> آدرس متغیرهایی است که باید از ورودی خوانده شوند و <عبارت 1> مشخص می‌کند مقادیر ورودی چگونه باید خوانده شوند و در متغیرهایی که آدرس آنها در <عبارت 2> مشخص شده است قرار گیرند. <عبارت 1> شامل سه نوع کاراکتر است.

۱) کاراکترهای فرمت ، این کاراکترها تعیین می‌کند که چه نوع اطلاعاتی باید از ورودی خوانده شوند. با ٪ می‌شوند. مثل d % برای خوانده اعداد صحیح از ورودی بکار می‌رود.

تابع getch ( )

این تابع ، کاراکتری را از ورودی خوانده و در متغیری قرار می‌دهد. این تابع در فایل conio.h قرار دارد. و نحوه کاربرد آن بصورت زیر است :

متغیر = getch ( ) ;

وقتی برنامه به این دستور ، می‌رسد ، منتظر می‌ماند تا کلیدی از صفحه کلید فشار داده شود. در اینصورت ، کاراکتری معادل آن کلید ، در متغیر قرار می‌گیرد. این تابع بصورت زیر نیز قابل استفاده است :

Getch ( ) ;

در اینصورت برنامه منتظر می ماند تاکلیدی از صفحه کلید فشار شود. پس از فشردن کلید ، اجرای بقیه دستورات برنامه ادامه می‌یابد.

اشاره‌گرها

اشاره‌گر متغیری است که آدرس متغیری دیگر را در خود نگهداری می‌کند. و به وسیله یک اشاره‌گر می‌توان هم به محتویات یک متغیر و هم به آدرس آن متغیر دسترسی داشت. اشارگرها در زبان برنامه نویسی C کاربردهای فراوانی داند. که از جمله می‌توان به تخصیص حافظه پویا و آرایه‌های پویا اشاره کرد. و آنها کار با رشته ها و آرایه‌ها و توابع را ساده‌تر می‌کنند و ارسال آرگومان از طریق فراخوانی با ارجاع را فراهم می‌کنند.

اشاره‌گر می‌تواند در متغیری ذخیره شود. اما ، اگر چه اشاره گر یک آدرس حافظه است و آدرس حافظه نیز یک عدد است. ولی نمی‌توان آن را در متغیری از نوع int یا double و غیره ذخیره کرد. بلکه متغیری که می‌خواهد اشاره‌گر را ذخیره کند باید از نوع اشاره باشد. این متغیرها را اشاره‌گر گویند. برای تعریف متغیرهای اشاره‌گر بصورت زیر عمل می‌کنیم :

نوع * متغیر ;

به این ترتیب برای تعریف متغیر اشاره‌گری که بخواهد آدرس متغیرهایی را نگهداری کند. باید نوع متغیر اشاره‌گر را همنوع با آن متغیرها در نظر گرفت و کنار متغیر اشاره‌گر ، علامت * را قـرار داد. بعنوان مثال دستور زیر را در نظر بگیرد.

Int * p;

این دستور را می‌توان بصورتهای زیر تفسیر کرد :

۱) p متغیر اشاره‌گری از نوع int است.

۲) P آدرس محلهایی از حافظه را که محتویات آنها مقادیری از نوع صحیح‌اند نگهداری می‌کند.

۳) P می تواند به محلهایی اشاره کند که محتویات آنها مقادیری صحیح می‌باشند.

اشاره‌گرها دارای دو عملگر & و * می‌باشند. دو عملگر & و * در انجام عملیات با اشاره‌گرها مورد استفاده قرار می‌گیرند هر یک از ایـن دو عملگـر ، یـک عملونـد دارنـد. عملگـر & آدرس عملونـد خودش را مشخص می‌کند و عملگر * محتویات جایی را مشخص می‌کند که عملوندش به آن شاره می‌کند.

متغیرهای پویا

چون اشاره‌گر می‌تواند آدرس محلی از حافظه را نگهداری کند. از طریق آدرس آن محل مـی‌توانـد محتویات آن محل را دستکاری کند. بنابراین لزومـی نـدارد آدرس محلـی کـه در اشـاره‌گـر قـرار می‌گیرد. دارای نام باشد. برای این منظور باید آدرس محلی از حافظه در اشاره‌گر قرار گیرد. امتیاز این روش این است که پس از اینکه کار با آن محل حافظه به اتمام رسید ، می‌تـوان آن حافظـه را آزاد کرد و به سیستم تحویل داد. این روش تخصیص حافظه پویا نام دارد.

می‌توان اینطور تصور کرد که محلهایی از حافظه ، که بدین طریق ، آدرس آنها در اشاره‌گر قرار می‌گیرند. ، متغیرهای بی نام هستند. این متغیرها را متغیرهای پویا نیز می‌گویند. زیرا در زمان اجرای برنامه توسط برنامه نویس ایجاد شده و از بین می‌روند.

تخصیص حافظه پویا

برای اخذ حافظه از سیستم ، از تابع ( ) malloc استفاده می‌شود. این تابع ، حافظه‌ای را از سیستم گرفته ، آدرس آن را در یک اشاره گر قرار می‌دهد. تابع ( )malloc در فایل alloc.h قرار دارد. بصورت زیر بکار می‌رود :

اشاره‌گر = (نوع) malloc (size) ;

در این روش کاربرد ، (نوع) اشاره‌گری است که آدرس حافظه باید در آن قرار گیرد و size میزان حافظه به بایت است و مشخص می‌کند که این تابع چند بایت از حافظه را باید از سیستم اخذ کند. اگر تابع ( ) malloc به هر دلیلی نتواند حافظه‌ای را از سیستم بگیرد و در اختیار برنامه نویس قرار دهد ، مقدار تهی (Null) را دراشاره‌گر قرار می‌دهد. اشاره‌گر تهی ، اشاره‌گری است که به جایی اشاره نمی‌کند. دستورات زیر را در نظر بگیرید.

```
Int * P ;
P = ( int * ) malloc ( sizeof ( int) ) ;
```

دستور اول ، P را اشاره‌گر صحیح تعریف می‌کند. و دستور دوم حافظه‌ای به اندازه sizeof (int) بایت را از سیستم گرفته آدرس آن را در P قرار می‌دهد. عبارت (int *) که قبل از ( ) malloc آمده است ، تبدیل نوع است.

برگرداندن حافظه به سیستم.

حافظه‌ای که بصورت پویا تخصیص یافته ، پس از استفاده باید به سیستم برگردانده شود. حافظه ای که به وسیله ( ) malloc اختصاص یافت با ( ) free به سیستم بر می‌گردد. ( ) Free در فایل alloc .h قرار دارد و بصورت زیر بکار می‌رود. :

free (اشاره‌گر) ;

توابع

در برنامه‌های طولانی و پیچیده که شامل چندین بخش منطقی و مستقل از هم هستند. بهتر است برای هر قسمت منطقی و مستقل ، برنامه‌ای جداگانه نوشته شود. برنامه‌ای که برای هر یک از بخشها نوشته می‌شود ، تابع نام دارد. در واقع تابع ، برنامه‌ای است که برای حل بخشی از مسئله نوشته می شود.

تعدادی از توابع که در اغلب برنامه‌ها مورد استفاده قرار می‌گیرند و کاربرد زیادی دارند از قبل نوشته شده ، به همراه کامپایلر ارائه می‌شوند. مثل توابع ( )sin و ( )cos که به ترتیب برای محاسبه سینوس وکسینوس زاویه بکار می‌روند یا تابع ( ) crscr که صفحه نمایش را پاک می‌کند. این توابع را توابع کتابخانه‌ای گویند.

هر تابع دارای دو جنبه است. جنبه تعریف تابع و جنبه فراخوانی آن ، جنبه تعریف تابع ، مجموعه‌ای از دستورات است که عملکرد تابع را مشخص می‌کند و جنبه فراخوانی تابع ، دستوری است که تابع را فراخوانی می‌کند. فراخوانی تابع با نام آن انجام می‌شود. نامگذاری برای تابع ، از

قانون نامگذاری برای متغیرها تبعیت می‌کند. توابع را باید پس از تابع ( ) main نوشت ، ساختار و اجزاء یک تابع در شکل زیر نمایش داده شده است.

```
< نوع تابع > نام تابع ( لیست پارامترها )
{
        دستور ۱
        دستور ۲
          ⋮
        دستور n
}
```

نوع تابع یکی از انـواع موجـود در C اسـت. مثـل double , float , int اگـر تابعـه‌ای بخواهـد مقداری را به تابع فراخوان برگرداند ، آن مقدار در نام تابع قرار می‌گیرد. چون هـر مقـداری دارای نوع است ، پس نام تابع نیز باید دارای نوع باشد. اگر تابع هیچ مقداری را به برنامـه فراخـوان خـود (برنامه‌ای که تابع در آن فراخوانی می‌شـود) برنگردانـد ، نـوع آن Void منظـور خواهـد شـد. پارامترها اطلاعاتی هستند که هنگام فراخوانی تابع ، از برنامه فراخوان به آن ارسال می‌شود.

بعبارت دیگر ، پارامترها وسیله‌ای برای تبادل اطلاعات بین تابع فراخوان و فراخوانی شوند هـستند. اگر تعداد پارامترها بیش از یکی باشد ، باید با کاما از هم جدا شوند. در لیست پارامترها نوع هر یک از پارامترها نیز مشخص می‌شود. اطلاعاتی که هنگام فراخوانی تابع ، در جلوی نام تابع (در داخـل پرانتز) ظاهر می‌شود ، آرگومان تابع نام دارند. ولی پارامترها متغیرهایی هستند که هنگام تعریـف تابع ، در جلوی نام تابع در داخل پرانتز قرار می‌گیرند. برای بکارگیری تابع در برنامه باید الگوی آن را در خارج از تابع main به کامپایلر اعلام کرد. الگوی تابع ، مشخص مـی‌کنـد کـه تـابع چگونـه فراخوانی می‌شود. الگوی تابع نیز به صورت زیر مشخص می‌شود :

; (لیست پارامترها) نام تابع < نوع تابع >

تابع چگونه کار می‌کند.

وقتی تابعی ، توسط تابع دیگر فراخوانی می‌شود. دستورات آن تابع اجرا می‌شوند. پس از اجرای دستورات تابع ، کنترل اجرای برنامه به برنامه فراخوانی بر می‌گردد. پس از برگشت از تابع فراخوانی شده ، اولین دستور بعد از فراخوانی تابع (در تابع فراخوان) اجرا می‌شود.

فصل ششم

تشریح و نوحی عملکرد برنامه

تشریح و نحوی عملکرد نرم افزار

کد نویسی این نرم افزار بر اساس محاسبات load flow به روش نیوتن – رافسون می‌باشد که کدهای نوشته شده ، این محاسبات را به زبان برنامه نویسی C ++ Builder تبدیل می‌کند. این برنامه شامل یک تابع اصلی (( ) main) و نه تابع فرعی می‌باشد که ، توابع فرعی در قابل اصلی فراخوانی می‌شوند. در واقع تابع ( ) main ، تابع فراخواننده ، و دیگر توابع تعریفی ، توابع فراخوانده شده توسط تابع ( )main می‌باشند. هر یک از توابع و ظیفه خاصی را بعهده داشته و قسمتی از مسئله را حل می‌کنند و به ترتیب نیاز فراخوانی می‌شوند. آرایه‌های پویای متعددی در این نرم افزار تعریف شده است که امکان تحلیل هر شبکه‌ای را با هر تعداد باس را فراهم می‌کند ،

و توضیحات زیر چگونگی و چرایی تعریف و کارکرد آرایـه‌هـا و دیگـر بخـشهای برنامـه را توصـیف می‌کند.

خطوط ۱ تا ۵ فایلهای کتابخانه‌ای نرم افزار را به برنامه پیوند می‌دهد. این فایلهای کتابخانه‌ای هـر کدام شامل توابعی‌ای خاص می‌باشد که توسـط کامپـایلر شـناخته شـده و از پـیش تعریـف شـده می‌باشد ، تا برنامه نویس در صورت لزوم بتوان آنها را در اختیار بگیرد. این فایلهـا در خطـوط زیـر دیده می‌شوند :

```
1: # gragma hdrstop
2 : # include < condefs.h
3: # include < stdio.h>
4: # include <conio.h>
5: # include <iostream.h>
```

در خطوط 6 تا ، پیش تعریف نه تابع دیده می‌شود ، که به قرار زیرند :

```
6: void get – array – Y (double ** , int) ;
7 : void get – arrag – teta ( double ** , int);
8 : double test ( double * , int , int );
9 : void inverse – matrix ( double ** , int , int );
10  void input – data ( bouble * , double * , double * , dable *, int *,int);
11  void full-pcal (double *,double*,double*;* double*;* int,int,double*);
12:void full-Qeal (double*,double*,double*,double**,int,int*,int,double*);
13  void full-del delta-v(double *, double*,double*,double*, int *,
double *;* int,int,int , double *);
```

14 void full-delP-Q (double *,double*,double*,double*,double*,double * , int * , int , int);

در خط 6 پیش تعریف تابعی با عنوان get-array-Y آمده است این تابع اندازه درایه‌های ماتریس Y_{bus} را در ورودی از کاربر دریافت می‌دارد و در آرایه‌ای بنام Y که در برنامه تعریف شده است ذخیره می‌کند. درخط 7 پیش تعریف تابعی با عنوان get – array – teta آمده است که این تابع تابع زاویه‌های ماتریس Y_{bus} را دریافت می‌دارد. در خط 8 پیش تعریف تابع تست دیده می‌شود این تابع شرط مربوط به محاسبات را که شامل تکرارهای متوالی تا convergc کردن سیستم می‌باشد را ایجاد می‌کند. خط 9 مربوط به پیش تعریف تابعی است. که ماتریس ژاکوبین را وارون می‌کند. خط 10 پیش تعریف تابعی دیگر با عنوان input-data می‌باشد که ، اطلاعات مربوط به نوع و مقادیر توانها و ولتاژهای باس بارها را از ورودی دریافت می‌کند. در خطوط 12, 11 پیش تعریف دو تابع با اسامی full-Pcal و full – Qcal دیده می‌شود همانطور که از اسامی این دو تابع پیداست full-Pcal توانهای محاسباتی باسها را در هر تکرار به جزء شین اسلک (شناور) انجام می‌دهد ، زیرا در شین اسلک توانهای تولیدی اکتیو و راکتیو را بعد از Convergc کردن سیستم محاسبه می‌کنیم. و تابع full – Qcal توانهای محاسباتی باسهایی را بدست می‌آورد که ولتاژ آن شین‌ها جزء مجهولات سیستم می‌باشند. در خط 13 پیش تعریف تابع full-del-delt-V دیده می‌شود. این تابع، تابعی است که ماتریس وارون ژاکوبین را در ماتریس حاصل از $\Delta Q_i, \Delta P_i$ ضرب کرده و نتایجی برای $\Delta|V_i|_{new}, \Delta S_{i\ new}$ بدست می آورد. خط 14 نیز پیش تعریف تابعی دیگر با عنوان full-del-P-Q می‌باشد. که وظیفه‌اش محاسبه ΔP_iها و ΔQ_i و قرار دادن آنها در ماتریس مربوطه می‌باشد.

در خط 15 دو متغیر عمومی تعریف شده است. این دو متغیر که دارای نامهای j,I مـی‌باشـند ، از آنجا که در تمامی توابع تعریف شده در برنامه استفاده می‌شوند متغیرهای عمومی تعریف شده‌انـد. متغیرهای عمومی به متغیرهایی اطلاق می‌شود که قبل از تـابع ( ) maind تعریـف مـی‌شـوند و برای تمامی توابع بکار رفته در برنامه قابل شناسایی می‌باشند و دیگر نیازی نیست در خـود توابـع هم تعریف شوند.

15 : int i,j ;

16 : # pragma argsused

خط 16 شامل دستور بیش پردازنده Pragma argsused می‌باشد و در خط بعدی تـابع اصـلی ( ) main وجود دارد. از اینجا به بعد کـار اصـلی برنامـه آغـاز مـی‌شـود و در بـدو امـری بایـست متغیرهای مورد نیاز تابع ( ) main و هم چنین اشاره‌گرهایی برای ایجاد آرایه‌های پویای مختلف تعریف شود. خط 18 آلولاد ({) مربوط به تابع ( ) main برای آغاز برنامه باز شده است.

17 : int main (int argc , char * argv [ ])

18 : {

19 : int A,K,L,n,n1,m=0, 0=T;

در خط 19 متغیرهای T, O,m,n1,n,L,K,A از نـوع int تعریـف شـده‌انـد کـه مقادیرصـحیح عددی را می‌توانند دریافت کنند. متغیر A در قسمتی از برنامه بکار می‌رود که توسط آن درایه‌های ماتریس ژاکوبین ($\frac{\partial Q_i}{\partial |V_i|}, \frac{\partial Q_i}{\partial S_i}, \frac{\partial P_i}{\partial |V_i|}, \frac{\partial P_i}{\partial S_i}$ ها) پر می‌شود که در بخش مربوط به آن پرداختـه خواهد شد. متغیرهای L,K نیز در حلقه‌های for استفاده می‌شـوند. امـا متغیـر n ، تعـداد بـاس بارهای شبکه مورد مطالعه را در خود ذخیره می‌کند. متغیر n1 مقـدار یکـی کمتـر n را در خـود

نگهداری می‌کند. متغیر m در همان ابتدای کار دارای مقدار صفر شده است. زیرا این متغیر می‌بایست تعداد شینهای ولتاژ مجهول ، یا بعارت دیگر شین‌هایی که ولتاژ آنها جزء مجهولات سیستم می‌باشد را شمارش نماید و در خود ذخیره نماید ، و از آنجا ممکن است برنامه بخشی از حافظه را به این متغیر نسبت دهد که دارای مقدار باشد ، به همین دلیل آنرا در ابتدا برابر با صفر قرار می‌دهیم. دو متغیر n1 , m برای اخذ حافظه از سیستم برای آرایه‌های ژاکوبین del-delta- V(J) و del-P-Q بکار می‌روند. متغیر 0 شمارنده تعداد تکرارها می‌باشد و T برای برقراری شرط حلقه while بکار می‌رود. خط 20 دو اشاره‌گر از نوع int با اسامی find , choose تعریف می‌کنند :

```
20  int * choose , * find;
```

choose اشاره‌گری است که ، حافظه‌ای به اندازه n بایت به اندازه int از سیستم دریافت می‌کند. در تابع input-data برای تعیین معلومات باس ها پنج گزینه برای انتخاب وجود دارد. گزینه اول باس اسلک ، گزینه دوم باس load ، گزینه سوم باس control ، گزینه چهارم باس generator و بالاخره گزینه پنجم باس Trans می باشد که کاربر می‌تواند هر یک از بر اساس ساختار شبکه مورد مطالعه خویش انتخاب نماید. شماره هریک از گزینه‌ها ، رمز آن گزینه می‌باشد. بعنوان مثال رمز باس اسلک شماره یک می‌باشد. هرگاه کاربر یکی از شماره‌های یک تا پنج را وارد کند ، در اینصورت همان شماره وارد خانه متناظر در آرایه choose قرار می‌گیرد. به این ترتیب آرایه Choose مشخص می‌کند که چه باسی اسلک ، generator , control , load و یا trans می‌باشد. بعبارت دیگر از خانه سوم آرایه Choose مقدار عددی دو را دانسته باشد. در اینصورت

می‌توان گفت که شین شماره دو شبکه مورد مطالعه از نوع load bus می‌باشد. از آنجا که آرایه‌های زبان برنامه نویسی C و C++ ، دارای اولین خانه با شماره صفر می‌باشد ، بنابراین ، خانه سوم chools ، شین دوم شبکه و خانه پنجم Choose ، شین سوم شبکه خواهد بود. آرایه Choose در فرایند این برنامه کاربردهای دیگری نیز دارد که در بخش‌های مربوط به آنها پرداخته خواهد شد.

اشاره‌گر find ، اشاره‌گری است که به اندازه متغیر m از سیستم حافظه اخذ می‌کند. همانطور که ذکر شد ، m متغیری است که تعداد شین‌های ولتاژ مجهول را در خود ذخیره می‌کند. بنابراین find آرایه‌ای است که به تعداد شین‌های ولتاژ مجهول خانه‌های حافظه را در بر می‌گیرد و به ترتیب شماره این شینه‌ها از کوچک به بزرگ در خانه‌های آن قرار می‌گیرد. آرایه find برای مشتقات جزئی Q نسبت به V,S و همچنین P به V کاربرد داشته و همچنین در توابع fuul-del-P-Q و full-del-delta-V نیز استفاده می‌شود ، که در مورد آنها نیز به تفصیل بحث خواهد شد.

خط 21 اشاره‌گرها PG,QL,PL , QG , V , delta از نوع double تعریف شده‌اند که هر کدام برای توان اکتیو بار ، توان راکتیو بار ، توان اکتیو و راکتیو تولیدی ، ولتاژ و زاویه ولتاژ شینه‌ها تعریف شده است. در آخر متغیر E برای ذخیره شرط محاسبات از نوع double در نظر گرفته شده است.

21 : double * PL, *,QL ,* , *QG ,*V , E;

در خط 22 اشاره‌گرهای دو بعدی Y, J , teta به ترتیب برای اندازه درایه‌های Y_{bus}، زاویه درایه‌های Y_{bus} و همچنین ماتریس ژاکوبین از نوع double تعریف شده‌اند.

22 : bouble ** Y,*teta ,*J;

خط 23 به ترتیب اشاره‌گرهای Pcal , Qcal ، del-P-Q و del-delta-V و help1,help2,help3 را از نوع double تعریف می‌کند. که در آن Pcal برای توان اکتیو محاسباتی شینه‌ها و Qcal برای توان راکتیو شینه‌ها و del-P-Q برای ماتریس حاصل از ΔP_i ها و ΔQ_i ها و del-delta-V برای ماتریس حاصل از $\Delta|V_i|, \Delta S_j$ ها لازم است و اشاره‌گرهای help1,help2,help3برای ایجاد آرایه‌های کمکی در توابع full-Pcal ، full-del-delta-V ، full – Qcal بکار می‌روند که در مبحث مربوط به آنها نیز پرداخته خواهد شد.

23 double * Pcal , * Qcal , *de-P-Q, *del delta-V,*help1,*help,* help3;

خط 24 پیغامی را مبنی بر اینکه : « این برنامه نرم افزاری برای پخش بار به روش نیوتن رافسون می‌باشد » را بعنوان اولین پیغام در خروجی چاپ می‌کند و خط بعدی پیغام دیگری که تقاضا می‌کند تا تعداد شینه‌های شبکه خود را وارد نمایید. در خط 26 این تعداد باس خوانده شده و در متغیر n قرار می‌گیرد. خط 27 نیز پیغام دیگری را چاپ کرده و اعلام می‌دارد تا شرط مربوط به محاسبات خود را وارد نمایید. (معمولاً عدد 0.001 می‌باشد) و در خط 23 این مقدار خوانده شده و در متغیر E ذخیره می‌شود.

24 : cout <<"\n\n\t * (this program is a software for load flow in nioton rafson method) * \n" ;

25 : coat <<"please entre number of bus : ";

```
26  cin>>  n;
27 :cout <<"\n please enter value for condition of calculations:";
28 : cin >> E ;
```

خطوط 29 تا 41 برنامه به ترتیب حافظه‌ای به اندازه n بایت برای آرایه‌های teta, Y,delta,V,QG,PG,QL,PL,choose از سیستم اخذ می‌دارد.

```
29 : choose = ( int * ) malloc ( n* size of ( int));
30 : PL = (double *) malloc ( n* sizeof ( double));
31 QL = (double*) malloc (n*sizeof (double));
32  PG = (double*) malloc (n*sizeof(double));
33  QG = (double*) malloc (n*sizeof (double));
34 : V = (double*) malloc ( n*sizeof(double));
35 : delta = (double*)malloc(n*sizeof(double));
36 : Y = ( double**) malloc (n*sizeof (double*));
37 : for (i=0 ; i<n ; i++)
38* (  Y+i)=(double*) malloc (n*sizeof(double));
39 :teta= (  double**)malloc (n*sizof(double*));
40  for (I = 0; i<n ; i++)
41 : * ( teta+i  )=(double*)malloc (n*sizeof(double));
```

خط 42 پیغامی را چاپ می‌کند تا کاربر ی کلید ، از صفحه کلید فشار دهد تا مقادیر اندازه درایه‌های Y_{bus} را وارد نماید. خط 43 دستور getch ( ) می‌باشد. که عمل فوق را انجام می‌دهد. درخط 44 تابع get-arrag-Y با دو آرگومان n,y فراخوانی می‌شود. این تابع وظیفه دریافت

مقادیر اندازه درایه‌های Y_{bus} را بعهده دارد. خطوط 47,46,45 همین کار را برای ماتریس زاویه درایه‌های Y_{bus} انجام می‌دهد.

```
42 : caut <<"\n* (please press any key to enter Ybus)*\n";
43 : getch ( ) ;
44 : get – array – Y (Y,n);
45 : caut << "\n *(please press any key to enter Y bus degree)*\n";
46 : getch ( ) ;
47 : get – array – teta( teta,n) ;
```

خطوط 48 تا 59 مقادیر آرایه V را برابر با یک و آرایه‌های QL,PL , PG , delta را برابر با صفر قرار می‌دهد. از آنجا که مقدار اولیه برای ولتاژهای مجهول یک می‌باشد ، بنابراین قبل شروع محاسبات بهتر آن است که تمامی ولتاژهای شینه‌ها یک شود. و اگر شینی دارای ولتاژ معلوم بود در تابع input – data مقدار آن از یک به مقدار جدید تغییر خواهد کرد و برای delta ها نیز ، از آنجا که مقدار اولیه تمامی زاویه ولتاژها صفر می‌باشد و زاویه ولتاژ شین slack نیز صفر است ، می‌بایست همگی صفر شوند.

```
48 : for ( i=0; i<n ;i++)
49  {
50 : * ( V+i) = 1 ;
51 : * ( deltati) = 0 ;
52 : * ( PG +i) = 0;
53 : * ( QG +i)= 0;
54 : * ( PL +i) = 0 ;
```

```
55 : * ( QL +i) = 0;
56 : }
```

در خط 57 تابع inpat – data فراخوانی می‌شود. این تابع شماره رمز هر شین را (این شماره رمزها همانطور که ذکر شد. عدد از ۱ تا ۵ می‌باشد) در آرایه Choose قرار می‌دهد. در این تابع ساختار ( ) swich بگونه‌ای بکار رفته است که با انتخاب هر کدام از گزینه‌های موجود در case مربوط اجرا شده و اطلاعات متناسب با آن شین در کاربر تقاضا می‌شود. در مورد تابع input-data بعداً به تفصیل سخن گفته خواهد شد.

```
57 : input – dat ( V,PL , QL , PG , choose , n);
```

خط 58 برای متغیر n1 مقدار تعیین می‌کند. این مقدار از مقدار ذخیره شده در n یکی کمتر می‌باشد.

```
58 : n1 = n – 1;
```

از خط 59 تا 63 کدها بگونه‌ای نوشته شده اند که ، برنامه به دنبال شینه‌هایی می‌گردد که ولتاژ آن شینه‌ها جزء مجهولات شبکه مورد بحث قرار دارد. دستورات آن بصورت زیر است.

```
59 : for ( U= 0 ; i<n ; i++)
60 : {
61 : if ( * choose +i) = = 2|| * ( choose + I ) = = S)
62 : m ++ ;
63 : }
```

همانطور که ملاحظه می شود حلقه for مربوطه ، تمامی شینه‌ها را از شماره صفر (اسلک) تا آخرین شماره (n-1) مرور می‌کند. اگر هر کدام از خانه‌های choose دارای مقدار 2 یا 5 باشد ،

تحت این شرایط یکی به مقدار m اضافه می‌شود. اگر بخاطر داشته باشد ، مقـدار m را در هنگـام تعریف صفر کرده بودیم چرا که ، اگر چنین نمی‌کردیم ممکن بود برنامه جایی از حافظـه را بـرای m اشغال نماید که آنجا دارای مقدار باشد. بنابراین عاقلانه آن بود که m در ابتدا صفر شود.

همانطور که قبلاً نیز عنوان شد. عدد 2 رمز شین‌های از نوع load و عدد 5 رمز شینِهـای trans می‌باشد. و این دو نوع شین هر دو جزء شینهای دارای ولتاژ مجهولند بنابراین پس از خاتمه حلقـه فور (for) ، عددی که در متغیر m قرار دارد ، معرف تعداد شین‌های دارای ولتاژ مجهول است.

متغیر m.n1 برای تخصیص برخی از حافظه‌ها و حلقه‌های for که در ادامه برنامه وجـود دارد بـه وفور بکار می‌رود.

64 : find = (int *) malloc (m* sizeof (int));

خط 64 به اندازه متغیر m برای آرایه find حافظه در نظر می‌گیرد. ارایه find در خطوط پائین‌تر صاحب مقدار خواهد شد. این مقادیر ، بگونه‌ای لحاظ می‌شوند که شماره شین‌های ولتاژ مجنول به ترتیب از عدد کمتر تا عدد بیشتر در آن مرتب شود. باید دانست زاویه ولتاژها برای تمامی شین‌هـا به جز شین اسلک مجهول است و این مسئله نظم خاصی را برای آن در نظـر مـی‌گیـرد امـا بـرای ولتاژهای مجهول این مسئله صادق نیست ، چرا که برنامه می‌بایست دربیابد که کدام شین‌ها ولتاژ مجهولند و یا بعبارت دیگر کدام شین‌ها دارای رمز دو (load) و یا رمز پنج (trans) می‌باشد. این درک توسط آرایه find به برنامه بخشیده می‌شود. زمانی که می‌خواهیم مشتقات جزئی $\frac{\partial P_i}{\partial S_i}$ هـا را محاسبه کنمی مشکل خاصی وجود ندارد ، زیرا همانطور که گفته شد S_iها برای تمـامی شـین

ها به جزء شین اسلک مجهول است و منظم می‌باشد اما برای محاسبه مشتقات جزئی $\frac{\partial P_i}{\partial |V_i|}$ و یا $\frac{\partial Q_i}{\partial |V_i|}$ و یا $\frac{\partial Q_i}{\partial S_i}$ برنامه توسط آرایه find در می‌یابد که مشتق چه چیزی را نسبت به چه چیز دیگری باید محاسبه نماید. بنابراین انجام محاسبات و تشکیل ژاکوبین و دیگر موارد که بعداً ذکر می‌شود ، بدون آرایه find امکان پذیر نمی‌باشد.

65 : del‡P-Q = (double*) malloc ((n1+m)*sizeof (double));

خط 65 آرایه دیگری تحت عنوان del-P-Q را به اندازه (n1+m) از حافظه دریافت می‌دارد. چرا؟ del-P-Q آرایه‌ای است که در محاسبات پخ بار به روش نیوتن رافسون بعد از محاسبه درایه‌های آن که شامل ΔP_i ها و ΔQ_i ها می‌باشد باید در ژاکوبین ضرب شود تا جوابی برای $\Delta |V_i|, \Delta S_i$ها بدست آید. می‌دانیم که n1 یکی کمتر از n می‌باشد ، یعنی همان تعداد زاویه ولتاژهای (Sها) مجهول و m به تعداد ولتاژهای مجهول شبکه مقدار می‌پذیرد و از آنجا که به اندازه S_i های مجهول ΔP_i داریم و به اندازه V_iهای مجهول (همان m) ΔQ_i داریم ، بنابراین می‌بایست اندازه آرایه del-P-Q به اندازه (n1+m) باشد. این مسئله برای ماتریس ژاکوبین و همچنین ماتریس ΔS_i ها و $\Delta |V_i|$ها ، که همان آرایه del-delta-V می‌باشد نیز کاملاً صادق است. به همین دلیل در طول برنامه این آرایه‌ها به اندازه (n1+m) از سیستم اخذ گردیده است.

66 : help1 = (double*)malloc (n*sizcof (double));

67 : help2 = (double*) malloc (n* sizcof (double));

68 : help3 =(double*) malloc ((n1+m)*sizcof(double));

خطوط 66 تا 68 سه آرایه با عنوان help1 , help2 , help 3 از حافظه اخذ کرده است. help1 , Help2 به ترتیب در توابع full-Pcal و full –Qcal بکار می‌رود از آنجا که Qcal , Pcal آرایه‌هایی به اندازه n خانه می‌باشد. help2,help1 نیز می بایست دارای سایزی به اندازهn (تعداد باس) داشته باشند. و چون help3 در تابع full-del-delt-V بکار می‌رود و چون این تابع به اندازه (n1+m) خانه دارد ، بنابراین help3 نیز می‌بایست (n1+m) خانه داشته باشد. این سه آرایه (help3,help2,help1) آرایه‌هایی کمکی برای توابع خود می‌باشند تا مقادیر محاسبه شده Qcal,Pcal و del-delta-V حاصل از تکرار قبلی را در خود محفوظ نگاه دارند. این مقادیر در تکرار آتی بکار می‌آیند.

```
69  for (i=0 ; i<n ; i++)
70 : {
71 : * (help1+i)=0;
72 : * ( help2+i)=0;
73 : }
74 : for (i=0; i<n1+m ; i++)
75 : *(help3+i)=0;
```

در خطوط 69 تا 75 مقادیر هر سه آرایه help3,help2,help1 را صفر می‌کند.

```
76 : L =0;
77 : for (i=0; i<n; i++)
78  {
79 : if (*(choos +i) = = 2 || * (choos+i  ) = =S)
80 : {
```

```
81* (   find +L)= I;
82 : L++;
83 : }
84  }
```

در خطوط 76 تا 84 مقادیر درایه‌های ماتریس یک بعدی choose را مرور می‌کند. و اگر choose دارای مقادیر 2 (رمز شین بار) و یا S (رمز شین انتقالی یا trans) باشد. آن شماره شین که همان مقدار متغیر iاست را در آرایه find قرار می‌دهد. به این ترتیب شماره شین‌های ولتاژ مجهول از عدد کمتر به سمت عدد بیشتر در find قرار می‌گیرد.

```
85 : del delta-V(double*)malloc((n1+m)*sizcof(double));
86 : Pcal=(double*)malloc(n*sizcof(double));
87  Qcal=(double*)malloc(n*sizcof(double));
```

در خطوط 85 تا 87 آرایه‌های Qcal , Pcal , del-delta-V را تعریف می‌کند.

```
88 : for (i=0; i<n1+m; i++)
89*(  del delta-V+i)=0;
90  for (i=0; i<n; i++)
91  {
92*(  Pcal+i)=0;
93*(  Qcal+i)=0;
94  }
```

در خطوط 88 الی 94 مقادیر Qcal , Pcal , del-delta-V را برابر با صفر قرار می‌دهد.

```
95  J=(double**) malloc ((n1+m)*sizcof(duble*));
96  for (i-0; , i<(n1+); , i++)
```

97* (J+i)=(double*) malloc ((n1+m)*sizecof(double));

خطوط 95 الی 97 ماتریس ژاکوبین را به اندازه (n1+m) تعریف می‌کند. از آنجا که n1 شین S که مجهول و m شین ولتاژ مجهول دارد بنابراین اندازه ژاکوبین باید (n1+m) سایز باشد.

تا اینجای برنامه هر چه که بیان تعریف آرایه‌های مورد نیاز برنامه و تعیین مقدار اولیه برای آرایه‌ها و متغیرها و تعریف آرایه‌های مورد نیاز توابع مختلف بود. از این به بعد هسته مرکزی برنامه شروع خواهد شد. و تکرارهای مکرر ، روش نیوتن رافسون تا مرحله برقراری شرط محاسبات آغاز می‌شود. در ابتدا می‌بایست مقادیر درایه‌های ماتریس ژاکوبین تعیین گردد و سپس ماتریس ژاکوبین وارون شود. بعد از آن مقادیر Pcal برای تمامی شین‌ها بجزء شین اسلک و Qcal برای شین‌های ولتاژ مجهول مشخص و محاسبه گردد. در مرحله بعدی نوبت به محاسبه ΔP_i ها برای تمامی شین به جز شین اسلک و محاسبه ΔQ_i ها برای شین‌های ولتاژ مجهول می‌رسد ، و سپس ماتریس del-P-Q پر شده و در ماتریس وارون ژاکوبین ضرب شود وتا مقادیر ΔS_iها و $\Delta|V_i|$ ها معین گردد. این فرایند آنقدر تکرار می‌شود تا شرط ارضاء شود هرگاه شرط ارضاء شد کنترل برنامه از حلقه while خارج شده و در انتها بقیه مجهولات محاسبه و در خروجی چاپ می‌شود.

تکرارهای محاسبات توسط حلقه do.....while انجام می‌گیرد. زیرا این حلقه ، حداقل برای یکبار تکرار می‌شود و شرط را در انتهای حلقه تست می‌کند ، چیزی نیازمان بوسیله آن مرتفع می‌شود.

خط 98 برنامه با دستور do{ آغاز می‌شود. از این دستور تا دستور while فرآیند برنامه مکرراً تکرار می‌شود بعد از دستور do بلافاصله ماتریس ژاکوبین در خطوط 99 تا 103 ، صفر می‌شود.

زیرا بعد از هر تکرار ، که درایه‌های ماتریس ژاکوبین از جمع چندین جمله شکل شده است ، اگر صفر نشود ، در تکرار آتی مقادیر جدید بدست آمده با مقادیر تکرار قبل جمعل شده و این موضوع باعث می‌شود تا نتیجه عمل غلط از آب درآید. باید دانست ، ماتریس ژاکوبین شامل چهار بخش مجزا از هم می‌باشد ، که هر بخش شیوه حل مسئله مخصوص بخود را دارد بعنوان مثال یک شبکه سه شینه با شین دوم ولتاژ مجهول دارای ژاکوبینی بصورت زیر است :

$$J=\left[\begin{array}{cc:c} \frac{\partial P_2}{\partial S_2} & \frac{\partial P_2}{\partial S_3} & \frac{\partial P_2}{\partial |V_2|} \\ \frac{\partial P_3}{\partial S_2} & \frac{\partial P_3}{\partial S_3} & \frac{\partial P_3}{\partial |V_2|} \\ \hdashline \frac{\partial Q_2}{\partial S_2} & \frac{\partial Q_2}{\partial S_3} & \frac{\partial Q_2}{\partial |V_2|} \end{array}\right] \begin{array}{l} \left.\begin{array}{l}\\\\\end{array}\right\} \text{n1} \\ \left.\begin{array}{l}\\\end{array}\right\} \text{m} \end{array} \Rightarrow \qquad J=\begin{bmatrix} J_1 & J_2 \\ J_3 & J_4 \end{bmatrix}$$

بنابراین برای تکمیل درایه‌های ژاکوبین نیاز به چهار نوع الگوریتم متفاوت داریم و هر بخش را بصورت مجزا در طور برنامه ظاهر می‌شود.

```
98  do {
99  for(i=0; , i<(n1+m); i++)
100: {
101: for (j=0; , j<(n1+m);j++)
102: *(J+i)+j)=0;
103: }
104: for (i=0; , i<n1; ,i++)
105: {
106: for (j=0; j<n1; j++)
107: {
```

```
108: if (i= =j)
109: {
110 : for (K=0; , k<n; k++)
111 : {
112: if (k! = (i+1))
113: A=1;
114: else
115: A=0;
116: *(*(J+i)+j)+= - (A*(*V+(i+1))*(*(V+k))*(*(*Y+(i+1))+k)* sin (*
delta +(i+1))-(*delta+k)-(*(*teta+( i+1))+k)));
117 : } // end of for (k)
118: } //end of if.
119 : else
120 : *(*J+1)j)=(*V(i+1))*(*V+(j+1)))*(*(*(Y+(i+1))+(j+1)))*sin (*
(delta+(i+1))-(*delta+(j+1)))-(*(*teta+( i+1)+(j+1))) );
121 : } // end of for (j)
122: } // end of for (i)
```

خطوط 104 تا 122 قسمت J_1 ماتریس ژاکوبین را تشکیل می‌دهد. حلقه for با شمارنده i نمایشگر سطرهای ماتریس ژاکوبین می‌باشد و حلقه for با شمارنده j نمایشگر ستونهای ماتریس ژاکوبین است. همانطور که عنوان شد قسمت J_1 ماتریس ژاکوبین دارای n1 سطر و n1 ستون است که از درایه صفر و صفر شروع می‌شود. بنابراین شمارنده‌های I,j می‌بایست از مقدار اولیه صفر شروع و تا n1 پیش روی کنند. قسمت J_1 ماتریس ژاکوبین مشتقات جزئی توانهای اکتیو نسبت به

زاویه ولتاژها است. حال اگر مشتق توان یک شین نسبت به زاویـه ولتـاژ همـان شـین باشـد مثـل $\frac{\partial P_2}{\partial S_2}$ ، دراینصورت $\frac{\partial P_2}{\partial S_2}$ از جمع چندین جمله حاصل می‌شود. دستور خـط 108 ایـن شـرط را ایجاد می‌کند. حال اگر شرط فوق اتفاق افتاده باشـد ، در یکـی از جمـلات در آرگومـان سینوسـی زاویه ولتاژ همان شین ظاهر نمی‌شود و جمله مربوط صفر خواهد بود. بنـابراین هرگـاه k (شـماره شین) با (i+1) برابر باشد جمله مربوطه صفر خواهد شد. چون مشتقات جزئی را برای شین اسلک در نظر نمی‌گیریم بنابراین شرط خط 112 باید k (شماره شین) نقـیض (i+1) باشـد بـا جملـه مربوطه صفر شود. حلقه for با شمارنده k برای وقتی است که مشتق توان یـک شـین نـسبت بـه زاویه‌های شین را محاسبه می‌کنیم. اما اگر مشتق توان یک شین را نسبت بـه زاویـه ولتـاژ شـینی دیگری محاسبه می‌کنیم ، جمله مربوط فقط یک جمله دارد ، چرا که در فرمول توان فقـط یکبـار زاویه ولتاژهای شینهای دیگر ظاهر مـی‌شـود. در اینـصورت جملـه بعـد از else خـط 119 اجـرا می‌شود.

```
123 : for (i=0; , i<n1; , i++)
124: {
125 : for (j=n1 , L=0 ; j<(n1+m); j++ ,L++)
126: {
127 : if (*(find+L) = = i+1)
128: {
129: for ( dk=0  ; k<n; k++)
130: {
131: if ( *(find +L) = = K)
```

```
132: A=2;
133: else
134: A=1;
135: *(*(J+i)j)+=A*(*(V+K)*(*(Y+(*(find+L)))+K))*cos ( * (delta+
(*(find+L)))-(*(delta+K))-(*(*teta+(*  find+L)))+k)));
136: } //end of for (k)
137: } // end of if (find)
138: else
139: * (*(J+i )+j)=(*(V+(i+1)))*(*(*(Y+(i+1))+(*(find+L))))*cos ( *
(delta+(i+1))-(*delta+(*find+L))))-(*(*(teta+( i+1))+(*(find+L)))));
140: }//end of for (j)
141: }//end of for (i)
```

خطوط 123 تا 141 قسمت J_2 یا همان $\frac{\partial P_i}{\partial |V_i|}$ ماتریس ژاکوبین را تشکیل می‌دهد. حلقه for با شمارنده i سطر و حلقه for با شمارنده j ستون ماتریس ژاکوبین را تشکیل می‌دهد. همانطور که گفته شد ، قسمت J_2 ژاکوبین از n1 شروع و به (n1+m) ختم می‌شوند.

بنابراین متغیر i باید از مقدار اولیه صفر شروع و به n1 ختم شود و متغیر j از مقدار اولیه n1 شروع و به (n1+m) ختم شود. متغیر L در حلقه for با شمارنده j واقع شده است و از مقدار اولیه صفر شروع می‌شود ، L برای آرایه find کار می‌کند. با افزایش ستونها (jها) در چرخشها مشتق گیری توانها نسبت به ولتاژهای مجهول از ولتاژ شین با عدد پائین به سمت شین با عدد بیش روی

می‌کند. بنابراین L که خانه‌های find را نشانه‌گیری می‌کند افزایش یافته و با افـزایش L مـشتق توانها نسبت به ولتاژی شین‌های بعدی حاصل می‌شود.

اگر مشتق توان به شین نسبت به ولتاژ همان شـین باشـد ، در اینـصورت جملـه حاصـله از جمـع چندین جمله تشکیل می‌شود. این شرط توسط دستور خط 127 ارضاء می‌شود. و اگر ایـن شـرط نایل آید حلقه for با شمارنده K وظیفه چرخش و ایجاد جملات مختلف را بعهده می‌گیرد. در این جملات همواره جمله‌ای وجود دارد که ولتاژ شین مربوط در عدد دو ضرب می‌شـود. ایـن شـرایط وقتی ایجاد می‌شود که در حلقه چرخش for با شمارنده K ، مقدار K با مقدار (find+L)* برابر شود. بنابراین ، هرگاه چنین شد متغیر A مقدار دو و در غیر اینصورت مقدار یک را می‌پذیرد. این شرط توسط خطوط 131 تا 134 ایجاد می‌شود. حال اگر مشتق توان یک شین (i+1) اندیس P است نسبت به ولتاژهای شین‌های دیگر باشد ، جمله حاصله فقط شـامل یـک جملـه مـی‌باشـد و دستور بعد از else خط 138 اجرا خواهد شد. بدین ترتیب توانستیم بخش دوم ماتریس ژاکـوبین را نیز ایجاد نماییم. در خطوط آتی به سراغ قسمتهای سوم (J_3) و چهارم (J_4) ماتریس ژاکوبین می‌رویم.

```
142: for (i=n1 ,L=0; i<(n1+m); i++ , L++)
143: {
144: for ( j=0; j<n1; j++)
145: {
146: if (*(find+L)= =j+1)
147: {
148: for (K=0;K<n; k++)
```

```
149: {
150: if (*(find+L)= = K)
151: A=0;
152: else
153:A=1;
154: *(*(J+i)+j)+=A*(*V+(*(find+L))))*(*(V+k))*(*(*(Y+(*(find+L )))
+K))*cos(*(delta+(*(find+L)))-(*(delta+K))-(*(*(teta+(*(find+L))) +K)));
155:}//end of for (K)
156: }//end of if (find)
157:else
158: *(*(J+i)+j)=-(*(V+(*(find+L)))*(*(V+(j+1))*(*(*(Y+(*(find +L )))
+(j+1)))*cos(*(delta+(*(find+L)))-(*(delta+(j+1))-(*(*(teta+(*( find
+L ))))+(j+1)));
159: }// end of for (j);
160: }//end of for (i);
```

خطوط 142 تا 160 قسمت سوم ماتریس ژاکوبین (J_3) را که $\frac{\partial Q_i}{\partial S_i}$ می‌باشد ، محاسبه می‌کند.

این قسمت از ژاکوبین از سطر n1 شروع و به سطر (n1+m) ختم می‌شود. و همچنین از ستون صفر شروع و به ستون n1 ختم می‌شود. بنابراین حلقه for با شمارنده iباید از مقدار اولیه n1 شروع و به مقدار (n1+m) ختم شود. و حلقه for با شمارنده j از مقدار اولیه صفر شروع و به مقدار n1 ختم می‌شود. زیرا iمعرف سطر و j معرف ستون در ماتریس ژاکوبین می‌باشد. متغیر L در حلقه for با شمارنده iاز مقدار اولیه صفر شروع می‌شود و با کام واحد در هر چرخش افزایش

می‌یابد. زیرا در این قسمت با افزایش سطر در ژاکوبین اندیس Q_i ها زیاد می‌شود. و از آنجا که تغییرات Q به V بستگی دارد ، بنابراین در سطرهای ژاکوبین Q_i های متناظر با V_iهای مجهول واقع می‌شوند. بدین ترتیب متغیر L که درآیه‌های آرایه find را نشانه می‌گیرد باعث می‌شود تا (find+L)* اندیسی برای Q_iها در مشتق گیری باشد. اندیس زاویه ولتاژها را (j+1) معین می‌کند. و از آنجا که زاویه ولتاژها برای تمامی شین‌ها به جزء شین اسلک مجهول است j را با یک جمع کرده‌ایم. حال اگر j+1 با مقدار ذخیره شده در (find+L)* یکی باشد ، به این معنی است که مشتق Q نسبت به زاویه ولتاژ شین خودش باید محاسبه شود. بنابراین جمله حاصل از جمع چندین جمله بوجود می‌آید و شرط خط 146 صادق شده و حلقه for با شمارندهK وظیفه ایجاد جملات مختلف را بعهده می‌گیرد. در بین این جملات ، یک جمله وجود دارد که زاویه ولتاژ آن ظاهر نمی‌شود و مشتق آن برابر با صفر خواهد شد. این شرط توسط دستور خط 150 ایجاد می‌شود.

اگر مشتق Q نسبت به زاویه ولتاژ شین‌های دیگری غیر از شین خودش باشد در اینصورت جلمه حاصل از مشتق گیری فقط یک جمله خواهد داشت. بنابراین دستور بعد از else خط 157 اجرا می‌شود. بدین ترتیب قسمت سوم (J_3) ژاکوبین نیز محاسبه و تعیین گردد. خطوط زیر قسمت چهارم (J_4) ژاکوبین را تعریف می‌کند.

```
161: for (i=n1 , L=0; i<(n1+m); i++ ,L++)
162: {
163: for (j=n1; , j<(n1+m); j++)
164: {
```

```
165: for (j=n1; j<(n1+m); j++)
166: {
167: for (k=0; k<n; k++)
168: {
169: if(*(find+L)= =K)
170: A=2;
171: else
172: A=1;
173: *(*(J+i)+j)+=A*(*(V+k))*(*(*(Y+(*(find+L)))+K))*sin ((*delta +
(*(find+L))))-(*(delta+k))-*(*(teta+(*find+L)))+K)));
174: }//end of for (k);
175: }// end of if
176: else
177: *(*(J+i)j+)=(*(V+(*(find+L))))*(*(*(Y+(*(find+L)))+(*find+(j-
n1))))*sin(*(delta+(*(find+L)))-(*(delta+(*(find+(j-n1)))))- (*(*(teta +
(*(find+L)))+(*(find+(j-n1)))));
178: }//end of for (j).
179: }//end of for (i).
```

خطوط 161 تا 179 قسمت چهارم ژاکوبین یا بعبارت دیگر $\frac{\partial Q_i}{\partial |V_i|}$ ها را محاسبه می‌نماید این قسمت از سطر و ستون n1 شروع و به سطر و ستون (n1+m) ختم می‌شود. بنابراین حلقه for با شمارنده iکه معرف سطر ژاکوبین می‌باشد با مقدار اولیه n1 آغاز و به (n1+m) ختم می‌شود و همچنین حلقه for با شمارنده j که معرف ستون ماتریس ژاکوبین است نیز از مقدار اولیه n1 آغاز

و به (n1+m) ختم می‌شود. متغیر L که درایه‌های آرایه find را نشانه‌گیری می کنـد در حلقـه for باشمارنده iمقدار اولیه صفر را می‌پذیرد و در هر چرخش این حلقه یکی بـه مقـدارش اضـافه می‌شود. مقادیر (find+L)* نمایشگر اندیس Q_i ها می‌باشد. که باید نـسبت بـه V_i هـا مـشتق گرفته شود. و مقادیر ((find(j-n1)* نیز نمایشگر اندیس V_iها می‌باشد. چرا که اندیس Q_i ها با افزایش سطر زیاد می‌شود و اندیس V_i ها با افزایش ستون زیاد می‌شـود. حـال اگـر مـشتق تـوان راکتیو شینی برحسب ولتاژ همان شین گرفته شود ، تحت این شرایط جواب حاصله از جمع چنـد جمله تشکیل می‌شود. در غیر اینصورت جواب فقط یک جمله دارد.این شرط در خط 165 وجـود دارد.

فرض کنید شرط فوق برقرار بوده باشد و مشتق توان راکتیو یک شین را نسبت به ولتاژ همان شین حساب می‌کنیم ، در اینصورت در جوابی که از چند جمله حاصل می‌شود جمله‌ای وجود دارد که ، ولتاژ همان شین مربوط در آن موجود است و دارای ضریب دو نیز می‌باشد. بنابراین برنامـه را بایـد بگونه‌ای نوشت تا این قضیه را درک نماید. برای دست یافتن به چنین هدفی در خـط 167 حلقـه for با شمارنده k چرخش جملات مختلف مربوط به ولتاژهای شین‌ها را ایجاد می‌کنـد و رد خـط 169 اگر (find+L)* که اندیس Q_i می‌باشد با K که اندیس ولتاژ شین‌ها می‌باشد یکـی شـد ، در اینصورت مشتق توان راکتیو نسبت به ولتاژ شین خودش می‌باشد ، که باید در ضریب دو ضرب شود. در غیر اینصورت ضریب یک خواهد بود. این ضریب در متغیر A واقع است. اگر مشتق تـوان راکتیو یک شین نسبت به ولتاژ همان شین مدنظر نباشد ، در اینصورت دستور بعـد از else خـط

176 اجرا می‌شود. بدین ترتیب خطوط 104 تا 179 ماتریس ژاکوبین بطور کامل پر کرده و تمام درایه‌های لازم را برای آن ایجاد می‌نماید.

در ادامه برنامه به خط 180 خواهیم رسید. در این خط تابعی با عنوان Inverse-matrix فراخوانی می‌شود. این تابع وظیفه وارون کردن ماتریس ژاکوبین را بعهده دارد.

180: inverse-matrix(J,n1,m);

خط بعدی تابع دیگر تحت عنوان full-Pcal فراخوانی می‌شود. وظیفه آن محاسبه توانهای اکتیو محاسباتی تمامی شین‌ها به جز شین اسلک می‌باشد. تابع دیگری در چند خط پائین‌تر وجود دارد با اسم full-Qcal این تابع وقتی فراخوانی می‌شود که m مخالف صفر است. این شرط در خط 182 موجود است. اگر به یاد داشته باشید m متغیری بود که تعداد شین‌های ولتاژ مجهول و یا بعبارت دیگر load و یا trans نباشد m برابر صفر است و آرایه find تشکیل نمی‌شود و به همین دلیل در ماتریس ژاکوبین دیگر $\frac{\partial Q_i}{\partial |V_i|}, \frac{\partial Q_i}{\partial S_i}$ وجود ندارد. بنابراین نیاز به محاسبه Qcal ها نمی‌باشد.

181: full – Pcal (Pcal,V,delta,Y,teta,n,0, help1);

182: if(m!=0)

183: {

184: full-Qcal(Qcal,V,delta,Y,teta,n, find,0,help2);

185: }

186: full-del-P-Q(del-P-Q,PG,PL,QG,QL,Pcal,Qcal,find,n,m);

187: full-del- delta-V(del-delta-V,del-P-Q, delta,V,find,J,n1,m,0,help3);

188: 0++;

189: if (test(delP-Q,n1,m)<E)

190:T=0;

191:else

192: T=1;

193: }while(T);

در خط 186 تابع full-del-P-Q فراخوانی می‌شود. این تابع مقادیر آرایه del-P-Q را محاسبه و پر می‌کند. مقادیر آرایه del-P-Q همان ΔP_iها و ΔQ_i ها می‌باشند. ودر خط 187 نیز تابع دیگری تحت عنوان full-del-delta-V فراخوانی می‌شود و این تابعع نیز مقادیر آرایه del-delta-V را محاسبه و پر می‌کنند ، مقادیر این آرایه همان ΔS_iها و $\Delta|V_i|$ها می‌باشد. در خط 188 به متغیر 0 که در هنگام تعریف صفر شده بود یکی اضافه می‌شود. این متغیر تعداد تکرارهای لازم برای converge کردن سیستم را ذخیره می‌کند. در خط 189 دستور ifی وجود دارد که در آن یک تابع به اسم test فراخوانی می‌شود تابع test در آرایه del-P-Q جستجو کرده و بزرگترین مقدار را از نظر قدر مطلق بر می‌گرداند. هرگاه مقدار جایگزین شده توسط تابع test کوچکتر از E که شرط محاسبات است شود. در این صورت سیستم converg کرده است و باید تکرارها خاتمه یابد.

هرگاه شرط if خط 189 برقرار شد در اینصورت متغیر T صفر می‌شود و حلقه while خاتمه می‌یابد در غیر اینصورت T برابر با یک است و تکرار دیگری آغاز می‌شود. از آنجا که شرط حلقه while تا زمانیکه برقرار باشد تکرار صورت می‌گیرد و ما می‌خواهیم که تا زمانیکه شرط برقرار نیست تکرار صورت گیرد. مجبور به تعریف یک if مطابق توضیحات فوق هستم. تا اینجا تمامی

تکرارها انجام می‌شود و وقتی از حلقه while خارج شویم به این معنی است که سیستم converge کرده است و نیازی به تکرار بعدی نیست. تا اینجا مقادیر ولتاژهای مجهول و زاویه ولتاژها بدست می آید.

بعد از این قسمت بقیه مجهولات سیستم از قبل توانهای اکتیو تولیدی شین‌ها و توانهای راکتیو تولیدی شین‌ها می‌بایست محاسبه شود و سپس جوابهای مسئله در خروجی چاپ شود.

```
194: for (i=0; i<n; i++)
195: {
196: * ( Pcal+i)=0;
197: *(Qcal+i)=9;
198: }
199: for (i=0; i<n; i++)
200: {
201: for (j=0; j<n; j++)
202: {
203: *(Pcal+i)+=(*(V+i)*(*(V+j))*(*(*(Y+i)+j))*cos(*( delta+i)- (*
(delta +j))-(*(*(teta+i )+j)) ));
204: *(Qcal+i)+=(*(V+i)*(*(V+j))*(*(*(Y+i)+j))*sin(*(delta+i)-
(*(delta+j)-(*(*(teta+i )+j))));
205: }
206: }
```

خطوط 194 تا 198 ارایه‌های Pcal , Qcal را صفر می‌کند. چون در خطوط بعدی این دو بگونه‌ای محاسبه می‌شوند که مقادیر قبلی با مقادیر فعلی جمع می‌شود ، برای پرهیز از این قضیه

می‌بایست آنها صفر شوند و مجدداً محاسبه شوند. خطوط 199 تا 206 نیز ایـن دو آرایـه را بطـور کامل محاسبه می‌نماید. البته با مقادیر جدید V و delta های جدید.

```
207: for(i=0; i<n; ,i++)
208: {
209: *(PG+i)=(*(Pcal+i))+(*(PL+i));
210: *(QG+i)=(*(Qcal+i))+(*(Ql+i));
211: }
```

خطوط 207 تا 211 توانهای اکتیو و راکتیو تولیدی تمامی شین‌ها را محاسبه می‌نماید.

```
212: for (i=0; i<n; i++)
213: *(delta+i)*=(180/3.141592654);
```

خطوط 212 تا 213 نیز زاویه ولتاژهای بدست آمده برحسب رادیان را به درجه تبدیل می‌کند.

```
214: printf ("\n\n\n")
215: printf("\t\t\t\t repeat is :%d",  0);
216: printf ("\n……\n")
217: printf ("\t bus\t vol \t\t deg\t\t PG\t\t QG");
218: printf ("\n……\n")
219: for (i=0; i<n; i++)
220: {
221: printf ("\t%d \t %lf \ t %lf \ t %lf \ t %lf",i+1,*(V+i),*(delta+i),*
(PG+i), *(QG+i));
222: printf ("\n\n");
223: }
```

```
224: printf ("\n……\n");
225: printf("\t bus\t PL\t\t QL");
226: printf( "\n……\n");
227:for (i=0; i<n; i++)
228: {
229: printf("\t %d /t %lf \t %lf",  i+l*(PL+i),*(QL+i));
230: printf ("\n\n");
231: }
```

خطوط تا 231 جوابهای بدست آمده از محاسبات نرم افزار را در خروجی استاندارد بطور مـنظم و هماهنگ چاپ می‌نماید.

```
232: for (i=0; i<n; i++)
233: {
234: free (*(Y+i));
235: free (*(teta+i));
236: }
237: for (i=0; i<(n1+m);i++)
238: {
239: free (*(J+i);
240: }
241: free (help1);
242: free (help2);
243: free (help3);
244: free(J);
```

```
245: free (V);
246: free (Y);
247: free(PL);
248: free (QL);
249: free(PG);
250: free(QG);
251: free(Pcal);
252: free(Qcal);
253: free(del delta-V);
254: free(teta);
255: free(delta);
256: free(delP-Q);
257: getch ( );
258: return 0;
259: : } //end of program.
```

خطوط بین 232 تا 256 کلیه حافظه‌های اخذ شده از سیتم را آزاد می‌کند. خط 257 باعث می‌شود. تا صفحه نمایش خروجی برای مشاهده نتایج عملیات نرم افزار ثابت باقی بماند و دستور خط 258 مقدار صفر را به سیستم عامل بر می‌گرداند ، زیرا تابع main( ) از نوع int تعریف شده بود و در نهایت برنامه در خط 259 خاتمه می‌پذیرد.

تا به حال هرچه که عنوان شد مربوط به دستورات موجود در تابع اصلی یا همان تابع main( ) بود. حال نوبت به آن رسیده که در مورد تابع فراخوانی شده درتابع mian( ) توضیحات لازم داده شود.

```
260: Void get-array-Y ( double ** Y,int n)
261: {
262: for (i=0; i<n; i++)
263: {
264: for ( j=0; j<n; j++)
265: {
266: printf ("\n please enter Value for Y [%d][%d]: ",i+1,j+1);
267: scanf ("%lf",  (*(Y+i)+j));
268: }
269: }
270: }
```

در خطوط بین 260 تا 270 تابعی با عنوان get-array-Y تعریف شده است. وظیفه این تابع دریافت مقادیر اندازه درایه‌های ماتریس Ybus شبکه مورد مطالعه می‌باشد این تابع ، در تابع main( ) در خط 4.4 فراخوانی می شود ، و پس از فراخوانی دستورات مربوط به آن اجرا خواهد شد. نوع تابع void است زیرا چیزی را به تابع فراخوان بر نمی‌گرداند.

```
271: void get-array-teta( double ** teta, int n)
272: {
273: double a;
274: for (i=0; i<n; i++)
275: {
276: for (j=0; j<n; j++)
277: {
```

```
278: printf ("\n please enter value for Ybus degree [%d][%d]:",i+1,j+1);
279: scanf 9"%lf", & a);
280: *(*(teta+i )+j)=(a*3.141592654)/180;
281: }
282: }
283: }
```

از خط 271 تا 283 تابع تحت عنوان get-array-teta تعریف شده است. وظیفه این تابع دریافت زوایای درایه‌های ماتریس Ybus شبکه می‌باشد. یک متغیر محلی در ساختار تابع از نوع double تعریف شده است که ، تابع مقادیر دریافتی را در آن قرار داده و سپس این مقادیر که برحسب درجه می‌باشد را به رادیان تبدیل و در آرایه teta قرار می‌دهد. علت این عمل ، بدین خاطر است که در زبان برنامه نویسی C توابع کتابخانه‌ای cos, sin آرگومانهای خود را بصورت رادیان می‌پذیرد. و جواب حاصل از محاسبات نیوتن رافسون برای زوایا نیز برحسب رادیان می‌باشد.

```
284: void input-data(double*V,double*PL,double QL , double* PG ,
{ ind * choose , int n}
285: {
286: for ( i=0; i<n; i++)
287: {
288: printf ("\n please choose on of the items below for bus (%d) :",i+1);
289: printf ("\n1) slack bus.\n");
290: printf ("\n2) load bus.\n") ;
291: printf ("\n3) control bus. \n");
292: printf ("\n4) genrator bus.\n");
```

```
293: printf ("\n5) trans bus.\n");
294: scanf ("%d , choos+i);
295: switch    (*(choos+i));
296: {
297: ease 1:
298: printf ("\n please enter value for voltag of slack bus:");
299: scanf ("%lf",V+i);
300: printf ("\n plese enter value for PL bus (%d):"i+1);
301: scanf ("%lf", PL+i);
302: Printf ("\n please enter value for QL bus (%d):"i+1);
303: scanf ("%lf",  QL+i);
304: break ;
305: case 2 :
306: printf ("\n please enter value for PL bus (%d):",i+1);
307: scanf("%lf",  PL+i);
308: printf ("\n please enter value for QL bus (%d):",i+1);
309: scanf ("%lf",QL+i);
310: break;
311: case 3 :
312: printf ("\n please enter value for PL bus (%d):"i+1);
313: scanf ("%lf",PL+i);
314: printf ("\n please enter value for QL bus (%d):" i+1);
315: scanf ("%lf",  QL +i);
```

```
316: printf ("\n please enter value for voltag of bus (%d):"i+1);
317: scanf ("\%lf",  V+i);
318: break ;
319: case 4 :
320: printf ("\n please enter value for voltag of bus (%d):",i+1);
321: scanf("%lf",V+i);
322: printf ("\n please enter value for PG bus (%d):" ,i+1);
323: scanf ("%lf",  PG+i);
324: printf ("\n please enter value for PL bus (%d) :" ,i+1);
325: scanf ("%lf",  PL+i);
326: printf ("\n please enter value for QL bus (%d):",i+1);
327: scanf ("%lf",  QL +i);
328: break ;
329: } // end of switch.
330: } // end of for.
331: }
```

تابع فوق که از خط 284 شروع و به خط 331 ختم می‌شود. و وظیفه دریافت اطلاعات مربوط به شین‌های شبکه مورد مطالعه را بعهده دارد. این تابع از نوع void است. زیرا هیچ مقداری را به تابع فراخوان بر نمی‌گرداند. آرایه‌ها و متغیرهای مورد نیاز تابع توسط آرگومانهای آن به تابع ارسال شده است. کل ساختار تابع در یک حلقه for قرار دارد که به تعداد باس‌های شبکه چرخش می‌کند تا بتواند اطلاعات مربوط به تمامی شین‌ها را دریافت کند. در ابتدای اجرای تابع یک پیغام به همراه پنج گزینه برای انتخاب در صفحه نمایش ظاهر می‌شود. کاربر می‌تواند طبق شبکه مورد نظر یکی

از موارد را طبق شماره‌های آن انتخاب نماید. هر شماره‌ای که انتخاب شود ، در خانه مربوط آرایه choose ذخیره می‌شود. و این‌آرایه (choose) در آرگومان دستور switch واقع شده است. و هر شماره‌ای را که کاربر انتخاب کرده باشد ، مطابق با آن ، case مورد نظر اجرا شده و اطلاعات مربوط به همان نوع شین از کاربر تقاضا می‌شود. این فرآیند برای تمامی شین اتفاق می‌افتد تا حلقه for خاتمه یافته و اطلاعات مربوط به تمامی شین‌ها مطابق با نوع آنها از کاربر دریافت شود. این تابع در خط 57 فراخوانی شده است.

323: void inverse-matrix (double ** J, int n1 , int m)

233: {

در خط 232 تعریف تابع inverse-matrix دیده می‌شود. این تابع از نوع void بوده بنابراین هیچ مقداری را به تابع ( )main (تابع فراخوانی) باز نمی‌گرداند. زیرا نیازی به این کار نداریم. وظیفه تابع وارون‌گیری از ماتریس ژاکوبین است. ماتریسی که (n1+m) سطر و ستون دارد. بنابراین آرایه و متغیرهای مورد نیاز آن که همان آرایه J (ژاکوبین) و متغیرهای m,n1 است به تابع ارسال می‌شود.

برای وارون نمودن ژاکوبین از روشی استفاده شده است که در محاسبات عددی به آن پرداخته می‌شود. این روش که با عنوان روش کیلی – هامیلتون از آن یاد می‌شود. راه حلی توانمند ، برای وارون‌گیری از ماتریس‌های مختلف می‌باشد. همانطور که قبلاً نیز ذکر گردید. فرمول لازم برای آن بصورت زیر است :

$$A^{-1} = -\frac{1}{P_n}\left[A^{n-1} + P_1 A^{n-2} + P_2 A^{n-3} + \ldots\ldots + P_{n-2} A + P_{n-1} I\right]$$

فرمول لازم برای P_K ها بصورت زیر است :

$$P_K = -\frac{1}{k}\left(S_K + P_1 S_{k-1} + P_2 S_{k-2} + \ldots\ldots\ldots + P_{n-1} S_1\right)$$

و S_i ها مجهول قطر اصلی ماتریس به توان i می‌باشد.

در اولین گام تابع می‌بایست S_i ها محاسبه نماید. در گام دوم نوبت به بدست آوردن P_k می‌رسد.

سپس بخش زیر از فرمول معکوس گیری انجام می‌گیرد.

$$\left(A^{n-1} + P_1 A^{n-2} + P_2 A^{n-3} + \ldots\ldots\ldots P_{n-2} A\right)$$

در مرحله بعد مقدار به جمله فوق اضافه شده و در نهایت کل جمله معادل بر مقدار $-P_n$ تقسیم می‌شود تا بدین ترتیب وارون ماتریس بدست آید.

حال مراحل فوق را به دستوران C تبدیل می کنیم :

```
334 : int K , a,t,L=0;
335: double **B, **W, **mat,**inv ,*P,*S,A,b;
336: B=(double**)malloc ((n1+m)*sizeof(doble*)) ;
337: for (i=0; i<(n1+m); i++)
338: *(B+i)=(double*)malloc ((n1+m)*sizeof(double));
339: W=(double**)malloc  ((n1+m)*sizeof(double*));
340: for (i=0; ,i<(n1+m); i++)
341: *(w+i)=(double*) malloc ((n1+m)*sizeof(double));
342: mat=(double**)malloc((n1+m)*sizeof(double*));
343: for (i=0; i<(n1+m); i++)
344: *(mat+i)=(double*)malloc((n1+m)*sizeof(double*));
```

```
345: inv=(double**)malloc((n1+m)*sizeof(double*));
346: for (i=0; i<(n1+m); i++)
347: *(inv+i)=(double*)malloc((n1+m)*sizeof(double));
348: P=(double*)malloc((n1+m)*sizeof(double));
349: S=(double*)malloc ((n1+m)*sizeof(double));
```

خطوط 334 و 335 متغیرهای لازم و همچنین اشاره‌گرهای مورد نیاز تابع را تعریف می‌کند. از خط 236 تا خط 349 حافظه‌های مورد نیاز برای این اشاره‌گرها نیز تخصیص می‌یابد لازم به ذکر است که‌آرایه P جایگزین برای P_k ها و آرایه S ، جایگزین برای S_i های فرمول ریاضی می‌باشد.

```
350: for (i=0; i<(n1+m); i++)
351: {
352: * ( S+i)=0;
353: * (P+i)=0;
354: }
```

خطوط 250 تا 354 ، آرایه‌های P,S را برابر با صفر قرار می‌دهد.

```
355: for ( i=0; i<(n1+m); i++)
356: {
357: for (j=0; j<(n1+m); j++)
358: *(*(B+i)+j)=(*(*(J+i)+j));
359: }
```

خطوط 355 تا 359 آرایه J را در آرایه B قرار می‌دهد. در خطوط بعدی از B برای بتوان رساندن ماتریس ژاکوبین استفاده می‌شود.

```
360: for (t=0; t<(n1+m; t++)
361: {
362: for (i=0; i<(n1+m); i++)
363: {
364: for (j=0; j<(n1+m); j++)
365: {
366: if(i= =j)
367: {
368: *(S+L)+=*(*(B+i)+j);
369: } //end of if
370: }
371: }
372: L++;
373: for (K=0; K<(n1+m); K++)
374: {
375: for (i=0; i<(n1+m; i++)
376: {
377: for (j=0; j<(n1+m); j++)
378: {
379: *(*(W+K)+i)+=(*(*(B+K)+j))*(*(*(J+j)+i));
380:}
381: }
382: }
```

```
383: for (i=0; i<(n1+m); i++)
384: {
385: for (j=0; j<(n1+m); j++)
386: {
387: *(*(B+i)+j)=(*(*(W+i)+j));
388: *(*(W+i)+j=0;
389: }
390: }
391: } // end of for (t)
```

خطوط 360 تا 391 وظیفه بدست آوردن S_i ها در ماتریس به iهـا را دارد. بـدین ترتیـب کـه حلقه for با شمارنده t به تعداد سطر یا ستون ژاکوبین چرخش می‌کند تـا بتوانـد هـم مـاتریس ژاکوبین را از توان یک تا توان (n1+m) ایجاد کند (n1+m)اندازه سطر و ستون ژاکوبین است و هم در این رهگذر S_iها را بدست آورد. می‌دانیم که تعداد S_iها نیز برابر n1+m است.

خطوط 362 تا 371 مقادیر S_iها را حساب می‌کند و دستور شرطی خط 366 موجب می‌شـود تا قطر اصلی ماتریس در S_i جمع شودو. خط 372 یک واحد به مقدار L اضافه مـی‌کنـد تـا در گردش بعدی t دومین مقدار برای دومین درآیه S حاصل شود.

خطوط 373 تا 382 ماتریس ژاکوبین را به توان می‌رساند و از خط 383 تا 390 موجب می‌شود تا توان بدست آمده از تکرار جاری در آرایه B واقع شود و آرایه W صفر می‌شود تا بـرای تکـرار آتی با مقادیر قبلی (جاری) جمع نگردد. با قرار دادن W در B ، در خط 387 موجب می‌شود تا

در تکرار آتی از ضرب آرایه B در J یک توان بالاتر حاصل شود. بدین ترتیب با هر تکـرار مقـادیر S_iها بدست آمده و آرایه B ، ماتریس ژاکوبین به توان (n1+m-1) ….,3,2,1 را ایجاد نماید.

```
392: for (i=0,b=1; i<(n1+m); i++,b++)
393: {
394: for (j=I,a=0; j>-1; j-- , a++)
395: {
396: if ( j= =i)
397: A=1;
398: else
399: A=(*(P+(a-1));
400: *(P+i)+=(-1/b)*(A*(*(S+j)));
401: }
402: }
```

از خط 392 تا 402 مقادیری برای آرایه P_k بدست می‌آورد. همـانطور کـه عنـوان شـد فرمـول ریاضی آن بصورت زیر است :

$$P_k = -\frac{1}{K}\left(S_k + P_1 S_{k-1} + P_2 S_{k-2} + \ldots\ldots + P_{n-1} S_1\right)$$

بطور مثال برای یک ماتریس 4×4, P_k ها بصورت زیر است.

$$P_1 = -S_1$$

$$P_2 = -\frac{1}{2}(S_2 + P_1 S_1)$$

$$P_3 = -\frac{1}{3}\left(s_3 + P_1 S_2 + P_2 S_1\right)$$

$$P_4 = -\frac{1}{4}\left(S_4 + P_1 S_3 + P_2 S_2 + P_3 S_1\right)$$

در حلقه‌ها شمارنده iبرای P و شمارنده j برای S می‌باشد. متغیر b که از مقدار اولیه یک شروع می‌شود ، با گام واحد در حلقه for با شمارنده iزیاد می‌شود. b در واقع جایگزین k در فرمول معادل ریاضی‌اش می‌باشد. حلقه for با شمارنده j از مقدار اولیه iشروع می‌شود. زیرا ، همانطور که از فرمول ریاضی آن پیداست اولین جمله در جملات معادل P_k ها ، اولینS همواره دارای اندیس خود P می‌باشد ، بنابراین j که اندیسی برای S باید از مقدار اولیه iکه اندیسی برای P است آغاز شود. در خط 396 یک جمله شرطی به چشم می‌خورد. این شرط اگر j با iبرابر باشد مقدار A که ضریب جملالت است را برابر با یک در غیر اینصورت برابر با ((P+(a-1))* قرار می‌دهد. این بدان علت است که اولین S در جملات همواره ضریب یک دارد ، و ضریب Sهای بعدی به ترتیب از $P_1, P_2, P_3, \ldots\ldots$ تا P_{n-1} پیش روی می‌کند. اما متغیر a در حلقه for با شمارنده j از مقدار اولیه صفر آغاز و با هر تکرار یک گام زیاد می‌شود ، در حالی که خود j یک گام کم می‌شود. دلیل این امر نیز این است که در فرمول ریاضی معادل همواره اندیس sها از مقدار بیشترین به سمت کمتر ، کاهش می‌یابد در حالی که اندیس Pهای ضریب از مقدار کمترین به سمت بیشتر ، افزایش می‌یابد.

پیچیدگی این چند خط برنامه هرچند کوتاه موجب گیج کنندگی آن می‌شود اما ، با کمی دقت قابل درک خواهد بود.

403: if(*(P+(n1+m-1))==0)

```
404: {
405: printf("\n*(the inrerse of matrix is impossible)*\n";
406: getch( );
407: }
```

اگر آخرین درآیه آرایه P برابر با صفر شود. در این صورت ماتریس مربوطه آن وارون پذیر نیست. هرگاه چنین اتفاقی رخ دهد ، خطوط 403 تا 407 پیغامی مبنی بر اینکه ماتریس وارون پذیر نیست ، در خروجی چاپ می‌کند. لازم به ذکر است که آخرین درآیه ماتریس *(P+(n1+m-1),P است زیرا آرایه‌ها در C از خانه شماره صفر شروع می‌شود.

```
408: for (i=0; i<(n1+m); i++)
409: {
410: for (j=0; j<(n1+m); j++)
411: {
412: *(*(inv+i)+j)=0;
413: *(*(mat+i)+j)=*(*(J+i)+j);
414: *(*(B+i)+j)=*(*(J+i)+j);
415: }
416: }
```

از خط 408 تا 416 آرایه inv را صفر و مقادیر J را در B,mat قرار می‌دهد تا در مراحل بعدی برنامه برای استفاده در فرآیند وارون‌گیری از ژاکوبین استفاده شوند.

```
417: for (t=0,a=(n1+m-3); t<(n1+m-1); t++ , a--)
418: {
419: if (t= = (n1+m-2))
```

```
420: A=1;
421: else
422: AS=*(P+a);
423: for (i=0; i<(n1+m); i++)
424: {
425: for (j=0; j<(n1+m); j++)
426: {
427: *(*(mat+i)+j)*=A;
428: *(*(inv+i)+j)+=*(*(mat+i)+j);
429: }
430: }
431: if (t== (n1+m-2))
432: break;
433: for (k=0; k<(n1+m); k++)
434: {
435: for (i=0; i<(n1+m); i++)
436: {
437: for (j=0; j<(n1+m); j++)
438: {
439: *(*(W+K)+i)+=(*(*(B+K)+j))*(*(*(J+j(+i));
440: } // end of for (j).
441: } // end of for (i).
442: } // end of for (k).
```

```
443: for (i=0; i<(n1+m); i++)
444: {
445: for (j=0; j<(n1+m); j++)
446: {
447: *(*(B+i)+j)=*(*(W+i)+j);
448: *(*(mat+i)+j)=*(*(W+i)+j);
449: *(*(W+i)+j=0;
450: }
451: }
```

خطوط 417 تا 451 عمل زیر را که بخشی از فرمول وارون نمودن ماتریس ژاکوبین است را انجام می‌دهد.

$$\left[A^{n-1} + P_1 A^{n-2} + P_2 A^{n-3} + \ldots\ldots\ldots + P_{n-2} A\right]$$

برنامه از سمت راست یکی ، یکی جملات را محاسبه و با مقدار قبلی جمع می‌کند تا کل جمله حاصل شود همانطور که ملاحظه می‌شود. می‌بایست ماتریس ژاکوبین را تا توان یکی کمتر از (n1+m) که اندازه سطر و ستون آن است ، برسانیم ، بنابراین حلقه for با شمارنده t که وظیفه چرخش توانها را بعهده دارد تا (n1+m-1) دفعه تکرار می‌شود. متغیر a در حلقه for با شمارنده t باید از مقدار اولیه (n1+m-1) شروع شود. زیرا در اولین تکرار توان یک ژاکوبین (خود ژاکوبین) در دوتا مانده به آخرین درایه ، آرایه P ضرب می‌شود. ودر تکرارهای بعدی یک واحد به توان ژاکوبی اضافه و یک واحد از درایه P کم می‌شود بنابراین در حلقه for ++t , --a می‌شود ، و از آنجا که آرایه‌ها در C از خانه صفر شروع و به خانه n-1 ختم می شوند (در یک آرایه n

خانه‌ای) بنابراین ، متغیر a باید از n1+m-3 شروع شود. همانطور که دیده می‌شود آخرین تـوان برای ماتریس وارون شونده $\left(A^{n-1}\right)$دارای ضریب یک است. به همین دلیل اگر مقـدار متغیـر t بـه عددn1+m-2 رسید باید ضریب ژاکوبین به توان رسیده یک باشد. این شرایط در خطوط 419 تا 422 تعریف شده است.

خطوط 423 تا 430 نیز متغیر A را که اگر آخرین تکرار باشد ، برابر با یـک و در غیـر اینـصورت برابر با $(P+a)$* را در آرایه mat ضرب و نتیجه را در خود mat قرار می‌دهد. لازم به ذکر است که در تکرار اول آرایه mat همان مقادیر ژاکوبین را دارد. در خط بعدی مقدار آرایه mat با آرایـه inv جمع و نتیجه در inv ذخیره می‌شود. در تکرار اول مقادیر inv صفر است. ایـن عمـل باعـث می‌شود که آرایه mat در P ها ضرب شده و با مقادیر قبلی محاسبات جمع شود تا فرمـول شـکل گیرد. در هر تکرار آرایه mat یک توان بالاتر از مرحله قبلی ژاکوبین را می پذیرد. شرط خط 431 و 432 باعث می‌شود تا در آخرین تکرار ادامه برنامه انجام شود زیرا دیگـر نیـازی بـه آن نـداریم و محاسبات این بخش تکمیل می‌شود. خطوط 433 تا 442 ژاکوبین را به توان می‌رساند و در آرایـه W قرار می‌دهد. خطوط 443 تـا 451 نیـز W را در mat , B قـرار مـی‌دهـد و خـود W صـفر می‌شود.

```
452: for (i=0; i<(n1+m); i++)
453: {
454: for (j=0; j<(n1+m); j++)
455: {
456: if ( i! =j)
```

```
457: *(*(mat+i)+j)=0;
458: else if (i= =j)
459: (*(*(mat+i)+j)=*(P+(n1+m-2));
460: }
461: }
```

اگر به خاطر داشته باشید. آخرین جمله از فرمول ریاضی وارون گیری ، ضرب یک ماتریس واحد به اندازه ژاکوبین در یکی مانده به آخرین درایه آرایه P بود. بنابراین در قطر اصلی ماتریس واحد به جای عدد یک می‌بایست مقدار یکی به آخرین درایه P قرار گیرد. برای آرایه P یکی به آخرین درایه ، خانه (n1+m-2) است. چون آرایه‌ها در C از خانه صفر شروع می‌شوند. خطوط بین 452 تا 461 این عمل را انجام می‌دهد.

شرط خط 456 بنان می‌دارد که اگر عناصر غیر قطری آرایه mat در دست رشد ، آنها را صفر و اگر عناصر قطری در دسترسند آنها را برابر با یکی به آخرین درایه P قرار بده بدین ترتیب ، آخرین جمله از فرمول نیز تکمیل می‌شود. حال نوبت آن است که این جمله با جملات دیگر جمع شود و همگی بر منفی آخرین درایه P تقسیم شوند.

```
462: for (i=0; i<(n1+m); i++)
463: {
464: for (j=0; j<(n1+m); j++)
465: {
466: *(*(inv+i)+j)+=*(*(mat+i)+j);
467: *(*(inv+i)+j)*=(-1/*P +(n1+m-1)));
468: *(*(J+i)+j)=*(*(inv+i)+j);
```

```
469: }
490: }
```

دستورات خطوط 462 تا 470 باعث می‌شود تا آرایه mat که اکنون آرایه‌ای است شامل درایه‌ها صفر به جزء قطر اصلی با مقدار یکی مانده به آخرین عنصر آرایه P با آرایه inv که شامل مجموع حاصلضرب درایه‌ها P در توانهایی از ژاکوبین است جمع شود و نتیجه آن بر منفی آخرین درایه P تقسیم شود. در نهایت نتیجه وارون شده ماتریس ژاکوبین در خود ژاکوبین قرار می‌گیرد تا دیگر محاسبان انجام شود.

```
471: for (i=0; i<(n+m); i++)
472: free (*(B+i));
473: free (B);
474: for (i=0; i<(n1+m); i++)
475: free (*(mat+i));
476: free (mat);
477: for ( i=0; i<(n1+m); i++)
478: free (*( inv +i));
479: free (inv);
480: for (i=0; i<(n1+m); i++)
481: free (*(W+i));
482: free(W);
483: free(S);
484: free(P);
485: }
```

از خط 471 تا 485 تمامی حافظه‌های اخذ شده در تابع فوق را آزاد می‌کند.

در خطوط زیر کد نویسی تابع full-Pcal ملاحظه می‌شود و پی رو آن توضیحات مربوط بـه ایـن تابع آمده است.

```
486: void full-Pcal (double**Pcal , double*V,doule*delta,double**Y ,
double **teta   int n, int 0 , double * help)
487: {
488: for (i=1; i<n; i++)
489: {
490: for (j=0; j<n; j++)
491: *(Pcal+i)+(*(v+i))*(*(v+j)(*(*(*(Y+i)+j))*cos(*( delta+i)- (*(delta
+j))-(*(*(teta+i )+j)));
492: if(O = =0 )
493: {
494: for (i=0; i<n; i++)
495: *(help+i)=(*(Pcal+i));
496: }
497: else if (O>0)
498: {
499: for (i=0; i<n; i++)
500: {
501: *(Pcal+i)-=*(help+i);
502: *(help+i)=*(Pcal);
503: }
```

```
504: } // end of else
505: }
```

تابعع فوق توان اکتیو محاسباتی را بدست مـی‌آورد ایـن تـابع در خـط 181 فراخـوانی مـی‌شـود. عملیات اصلی این تابع در خط 491 ، درون تابع است. اگر تابع برای اولین بار فراخوانی شده باشد ، محاسبات را به درستی انجا می‌دهد. ولی اگر فراخوانی چندین مرتبه تکـرار شـود ، در اینـصورت مقدار محاسبه شده تکرار قبلی در Pcal باقی بود و مقدار جدید را نیز به آن اضافه می‌کـرد. بـرای جلوگیری از این عمل ، مقادیر تکرارهای قبلی را در آرایه‌های تحت عنوان help ذخیره کـردیم و مقادیر جدید Pcal را از help کم نمودیم تا مقدار واقعه‌ای تکرار آتی بدسـت آیـد. ایـن عمـل در خطوط 492 تا 504 انجام می‌شود.

با اندکی تفاوت تابع full-Qcal نیز الگـوریتمی مـشابه بـا full-Pcal دارد در زی بـه ایـن تـابع پرداخته می شود.

```
506: void full- Qcal(double*Qcal,double*V,double*delta,double**Y  ,
double**teta  int n , int *find , int 0 , double*help)
507: {
508: int L;
509: for ( i=1, L=0 ; i<n; i++)
510: {
511: if (*(find+L)==i)
512: {
513: for (j=0; j<n; j++)
```

```
514: *(Qcal+i)+= (v+i)*(*(v+j))*(*(*(Y+i)+j))*sin(*(delta+i-
(*( delta+j ))- (*(*(teta+i)+j)));
515: L++;
516: } // end of if
517: } // end of for (i)
518: if (O= =0)
519: {
520: for (i=0; i<n; i++)
521: *(help+i)=(*(Qcal+i));
522: }
523: else of (O>0)
524: {
525: for (i=0;i<n; i++)
526: {
527: *(Qcal+i)-=(*(help+i));
528: *(help+i)=(*(Qcal+i));
529: }
530: } // end of else if
531: }
```

تابع فوق توانهای راکتیو محاسباتی شین‌های ولتاژ مجهول را بدست می آورد. همانطور که ملاحظه می‌شود. دوباره آرایه find برای تشخیص اینکه کدام شین را باید در نظر گرفت و توان راکتیو آنرا حساب کرده مورد استفاده قرار گرفته است.

حلقه for با شمارنده i، iرا از مقدار اولیه یک شروع می‌کند زیرا با شین اسلک کاری نداریم. در این حلقه L که اندیس find است از صفر شروع می‌شود. اما L در حلقه for با شمارنده j افزایش پیدا می‌کند. (خط 515) به بدان علت است که اگر شرط خط 511 برقرار نبود مقدار L زیاد نشود و ثابت باقی بماند تا محتوی درایه Lام ارایه find با مقدار iبعدی (که همان شماره شین محسوب می‌شود) مقایسه شود. هرگاه شرط برقرار شود Qcal محاسبه خواهد شد. بدین ترتیب توانهای راکتیو فقط برای شین‌های ولتاژ مجنول بدست می آیند ، چیز مدنظر ما بود.

از خطوط 518 تا 530 همان کاری را انجا می‌دهد که در تابع full-Pcal انجام شده بود. یعنی برای اینکه مقادیر قبلی با مقادیر جاری Qcal با هم جمع نشود آرایه help را بعنوان مددگر Qcal تعریف کردیم تا مقادیر هر تکرار را محفوظ نگاهدارد و در تکرار بعدی Qcal را از آن کم می‌کنیم تا مقدار واقعی Qcal باقی بماند.

```
535: void full-del-P-Q (double*del-P-Q,double*PG,double*PL, double
*Pcal , double*QG , double*QL,double*Qcal,int*find,int n,int m)
533: {
534: int n1 , L;
535: n1=n-1;
536: for(i=0;i<n1; i++)
537: *(delP-Q )=(*(PG+(i+1)))-(*(PL+(i+1)))-(*(Pcal+(i+1));
538: for (i=n1 ,L=0 i<(n1+m); i++ , L++)
539: *(delP-Q+i )=*(QG+(*(find+L)))-(*(QL+(*(find+L))))-(* Qcal +
(*(find+L))));
540: }
```

تابع فوق مقادیر $\Delta Q_i, \Delta P_i$ را محاسبه و در آرایه del-P-Q قرار می‌دهد همانطور که ذکر گردید. ΔP_iها به اندازه n1 (یکی کمتر از تعداد باس) و ΔQ_i ها به تعداد m (تعداد شین‌های ولتاژ مجهول) می‌باشد. از آنجایی که شماره شین‌های ولتاژ مجهول در آرایه find به ترتیب وجود دارد.بنابراین از این آرایه بعنوان اندیس توانهای راکتیو استفاده شده است.

```
541: void full-del-deta-v(double*del delta0v,double*delP-Q,* double *
delta ,double*Y,int *find,double*, int n1 ,int m,
542: int O , double*help )
543: {
544: int L;
545: for(i=0; j<(n1+m);j++)
546: {
547: for (j=0; j<(n1+m); j++)
548: *(deldelta 0V+i)+=(*(*(J+i)+j))*(*(del-P-Q-j));
549: }
550: if (O= =0)
551: {
552: for (i=0;i<(n1+m);i++)
553: *(help+i)=*(del delta-v+i));
554: }
555: else if (O>0)
556: {
557: for (i=0; i<(n1+m); i++)
```

```
558: {
559: *(delдelta -v+t)- =(*(help+i);
560: *(help+i)=(*(del-delta0v+i));
561: }
562: } // end of else if.
563: for (i=0; i<n1; i++)
564: {
565: *(delta+(i+1)=(*(delta+(i+1))+(*(del-delta-v+i)));
566: }
567: for (i=n1,L=0 i<(n1+m);i++, L++)
568: {
569: *(V+(*(find+L)))=(*(V+(*(find+L)))+(*(delдelta 0V+i));
570: }
571: }
```

تابع فوق در خط 187 فراخوانی می‌شود. وظیفه این تابع پر کـردن مـاتریس del-delta-V کـه همان ΔS_iها و $\Delta|V_i|$ها می‌باشد و سپس بدست آوردن $S_{new}, |V|_{new}$ می‌باشـد. از خـط 545 تـا 549 تابع آرایه ژاکوبین (J) که اکنون وارون شده است را در آرایـه del-P-Q ضـرب مـی‌کنـد و نتایج این حاصلضرب را در آرایه del-delta-V قرار می‌دهـد. درایـه‌هـای del-delta-V همـان ΔS_i و $\Delta|V_i|$هامی‌باشد. از خط 550 تا 562 نیز برای اینکه مقادیر تکرار قبلی del-delta-V با مقادیر جاری جمع نشود از آرایه help مثل دو تابع قبل کمک گرفتیم. در مرحله آخـر ایـن تـابع

می‌بایست آرایه‌های delta , V به روز شوند. بنابراین درایه (i+1) آرایه delta را با درایه iآرایه del-delta-V جمع می‌کنیم تا جواب S_{new} شین دوم شبکه بدست آید.

همانطور که می‌دانیم آرایه‌ها در C از درایه صفر شروع می‌شوند و در کل برنامه شین اسلک برای نرم افزار شین شماره صفر و برای ماشین شماره یک بود. به همین دلیل در خط 565 از (delt+(i+1)) شروع کردیم چون زاویه ولتاژ شین اسلک معین و برابر با صفر بود ، و (del-delta-V+i) نیز به (delta +(i+1)) مربوط می شود چون آن نیز اولین درایه در ماتریس del-delta-V می‌باشد.

خطوط 567 تا 570 نیز ولتاژها را برای شینهای مجهول حساب می‌کند. همانطور که ملاحظه می‌شود ، چرخش حلقه برای آن از n1 شروع و به n1+m ختم می‌شود. یعنی m دفعه می‌چرخد ، که همان تعداد شین‌های ولتاژ مجهول است بدین ترتیب مقادیر مجهول برای ولتاژها و زاویه ولتاژها در این تابع معین می‌شود.

```
572: double test (double*delP-Q  , int n1 , int m)
573: {
574: double a;
575: a =-1;
576: for ( i=0; i<(n1+m); i++)
577: {
578: if (fabs (*(delP-Q-+i ))>a)
579: a=fabs (*(delP-Q+i ));
580: }
```

581: return (a);

582: }

تابع فوق که در بین خطوط 572 تا 582 واقع است. وظیفه پیدا کردن بزرگترین عدد از نظر قـدر مطلق را در آرایه del-P-Q بعهده دارد. همانطور که می‌دانیم باید تکرارهای روش نیوتن-رافسون را تا بدآنجایی انجام دهیم که بزرگترین عدد از نظر قدر مطلق در ماتریس $\Delta Q_i, \Delta P_i$ از شرط در نظر گرفته شده ε(معمولاً $\varepsilon = 0.0001$ در نظر گرفته می‌شود) کوچکتر شود. هرگاه چنین شد ، آنگاه باید تکرارها متوقف شود.

این تابع در خط 189 فراخوانی شده است و در آرگومان شرط if می‌باشد که هرگاه مقدار برگشتی این تابع کوچکتر متغیر E شد. آنگاه یک متغیر بنام T مقدار صفر می‌پذیرد ، و از آنجایی که T در آرگومان حلقه while قرار دارد بنابراین برای while دیگر شـرط برقـرار نبـوده و بـدین ترتیـب تکرارها پایان می‌پذیرد.

فصل هفتم

نرم افزار

```
#pragma hdrstop
#include <condefsh<
#include <stdio.h<
#include <conio.h<
#include<iostream.h<



void get_array_Y(double **,int;(
void get_array_teta(double **,int;(
double test(double *,int,int;(
void inverse_matrix(double **,int ,int;(
void input_data(double *,double *,double *,double *,int *,int;(
void full_Pcal(double *,double *,double *,double **,double **,int,int,double;(*
void full_Qcal(double *,double *,double *,double **,double **,int,int *,int,double
;(*
void full_del_delta_V(double *,double *,double * ,double *,int *,double
**,int,int,int,double;(*
void full_del_P_Q(double *,double *,double *,double *,double *,double *,double
*,int *,int,int;(

 int i,j;
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv([]
}


 int A,k,l,n,n1,m=0,o=0,T;
 int *choose, *find;
 double *PL,*QL, *PG, *QG, *V, *delta, E;
 double **Y,**teta,**J;
 double *Pcal,*Qcal,*del_P_Q,*del_delta_V,*help1,*help2,*help3;

  printf("\n\n\t*(This program is a software for load flow in Niotonrafson
method)*\n;("
  printf("\n Please enter number of bus;(":
  scanf("%d",&n;(
  printf("\n Please enter value for condition of calculations;(":
  scanf("%lf",&E;(


  choose=(int *)malloc(n*sizeof(int;((
  PL=(double *)malloc(n*sizeof(double;((
  QL=(double *)malloc(n*sizeof(double;((
  PG=(double *)malloc(n*sizeof(double;((
  QG=(double *)malloc(n*sizeof(double;((
  V=(double *)malloc(n*sizeof(double;((
  delta=(double *)malloc(n*sizeof(double;((
```

```
  Y=(double **)malloc(n*sizeof(double;((*
  for(i=0;i<n;i(++
)*  Y+i)=(double *)malloc(n*sizeof(double;((

  teta=(double **)malloc(n*sizeof(double;((*
  for(i=0;i<n;i(++
)*  teta+i)=(double  *)malloc(n*sizeof(double;((




  printf("\n *(Please press any key to enter Y bus)*\n;("
  getch;()
  get_array_Y(Y,n;(
  printf("\n *(Please press any key to enter Y bus degree)*\n;("
  getch;()
  get_array_teta(teta,n;(

  for(i=0;i<n;i(++
}
)*  V+i)=1;
)*  delta+i)=0;
)*  PG+i)=0;
)*  QG+i)=0;
)*  PL+i)=0;
)*  QL+i)=0;
{

 input_data(V,PL,QL,PG,choose,n;(
//---------------------------------------------------------------------------

  n1=n-1;
  for(i=0;i<n;i(++
}
    if( (*(choose+i)==2)||( *(choose+i)==5( (
    m;++
//{  end of for

  find=(int *)malloc(m*sizeof(int;((
  del_P_Q=(double *)malloc((n1+m)*sizeof(double;((

  help1=(double *)malloc(n*sizeof(double;((
  help2=(double *)malloc(n*sizeof(double;((
  help3=(double *)malloc((n1+m)*sizeof(double;((

  for(i=0;i<n;i(++
}
)*  help1+i)=0;
)*  help2+i)=0;
```

```
{

  for(i=0;i<(n1+m);i(++
)*  help3+i)=0;
-------------------------------------------------------------------------------//
  l=0;
  for(i=0;i<n;i(++
}
    if( *(choose+i)==2 || *(choose+i)==5(
}
)*      find+l)=i;
     l;++
{
{
-------------------------------------------------------------------------------//


  del_delta_V=(double *)malloc((n1+m)*sizeof(double;((
  Pcal=(double *)malloc(n*sizeof(double;((
  Qcal=(double *)malloc(n*sizeof(double;((

  for(i=0;i<(n1+m);i(++
)*  del_delta_V+i)=0;

  for(i=0;i<n;i(++
}
)*  Pcal+i)=0;
)*  Qcal+i)=0;
{

  J=(double **)malloc((n1+m)*sizeof(double;((*
    for(i=0;i<(n1+m);i(++
)*  J+i)=(double *)malloc((n1+m)*sizeof(double;((



-------------------------------------------------------------------------------//

 do}

  for(i=0;i<(n1+m);i(++
}
   for(j=0;j<(n1+m);j(++
)*)*    J+i)+j)=0;
{



//divid P to delta.
  for(i=0;i<n1;i(++
```

```
}
  for(j=0;j<n1;j(++
}
   if(i==j(
}
    for(k=0;k<n;k(++
}
      if( k!=(i+1( (

        A=1;
        else
        A=0;

)*)*J+i)+j)+=-( A* (*(V+(i+1))) * (*(V+k)) * (*(*(Y+(i+1))+k)) * sin
)*) )delta+(i+1))) - (*(delta+k)) - (*(*(teta+(i+1))+k;(( ((
//{  end of for(k.(
//{ end of if.

 else
}
)*)*J+i)+j)= (*(V+(i+1))) * (*(V+(j+1))) * (*(*(Y+(i+1))+(j+1))) * sin
)*) )delta+(i+1))) - (*(delta+(j+1))) - (*(*(teta+(i+1))+(j+1;( (((
{
//{  end of for(j.(
//{ end offor(i.(
-------------------------------------------------------------------------------//
//divid P to V.
 for(i=0;i<n1;i(++
}

  for(j=n1,l=0;j<(n1+m);j++,l(++
}

  if( *(find+l)==i+1(
}

   for(k=0;k<n;k(++
}
    if( *(find+l)==k(

     A=2;
    else
     A=1;

)*)*J+i)+j)+= A*(*(V+k))*(*(*(Y+(*(find+l)))+k))*cos
)* )delta+(*(find+l)))-(*(delta+k))-(*(*(teta+(*(find+l)))+k;( ((
//{  end of for(k.(
//{ end of if(find.(

 else
```

```
}
)*)*J+i)+j)=(*(V+(i+1)))*(*(*(Y+(i+1))+(*(find+l))))*cos
)* )delta+(i+1))-(*(delta+(*(find+l))))-(*(*(teta+(i+1))+(*(find+l;( ((((
{
//{  end of for(j.(
//{ end of for(i.(
-------------------------------------------------------------------------//
//divid Q to delta.

 for(i=n1,l=0;i<(n1+m);i++,l(++
}

  for(j=0;j<n1;j(++
}
   if( *(find+l)==j+1(
}

    for(k=0;k<n;k(++
}
     if( *(find+l)==k(

      A=0;
     else
      A=1;

)*)*J+i)+j)+= A*(*(V+(*(find+l))))*(*(V+k))*(*(*(Y+(*(find+l)))+k))*cos
)* )delta+(*(find+l)))-(*(delta+k))-(*(*(teta+(*(find+l)))+k;( ((
//{  end of for(k.(
//{ end of if(find.(

else
}
)*)*J+i)+j)=-( *(V+(*(find+l)))*(*(V+(j+1)))*(*(*(Y+(*(find +l)))+(j+1)))* cos
)* )delta+(*(find+l)))-(*(delta+(j+1)))-(*(*(teta+(*(find+l)))+(j+1;(( (((
{
//{  end of for(j.(
//{ end of for(i.(
-------------------------------------------------------------------------//
//divid Q to V.
 for(i=n1,l=0;i<(n1+m);i++,l(++
}

  for(j=n1;j<(n1+m);j(++
}
   if( *(find+l)==(*(find+(j-n1( (((
}
    for(k=0;k<n;k(++
}
     if( *(find+l)==k(
      A=2;
```

```
    else
     A=1;

)*)*J+i)+j)+= A*(*(V+k))*(*(*(Y+(*(find+l)))+k))*sin
)*) )delta+(*(find+l))))-(*(delta+k))-(*(*(teta+(*(find+l)))+k;( ((
//{  end of for(k.(
//{ end of if.

 else
}
)*)*J+i)+j)= (*(V+(*(find +l))))*(*(*( Y+(*(find +l)))+(*(find+(j-n1)))))*sin
)* )delta+(*(find+l)))-(*(delta+(*(find+(j-n1)))))-(*(*(teta+(*(find+l)))+(*(find+(j-
n1;( (((((
{
//{  end of for(j.(
//{ end of for(i.(

-------------------------------------------------------------------------------//

 inverse_matrix(J,n1,m;(

 full_Pcal (Pcal ,V ,delta ,Y ,teta ,n ,o,help1;(

 if(m!=0(
}
 full_Qcal (Qcal ,V ,delta ,Y ,teta ,n ,find ,o,help2;(
{
 full_del_P_Q (del_P_Q ,PG ,PL ,Pcal ,QG ,QL ,Qcal ,find ,n ,m;(

 full_del_delta_V (del_delta_V,del_P_Q ,delta ,V,find ,J ,n1 ,m,o,help3;(

 o;++

 if( test(del_P_Q,n1,m) < E(
 T=0;
 else
 T=1;

 {  while(T;(

-------------------------------------------------------------------------------//
for(i=0;i<n;i(++
}
)* Pcal+i)=0;
)* Qcal+i)=0;
{

for(i=0;i<n;i(++
}
 for(j=0;j<n;j(++
```

```
}
)* Pcal+i)+=(*(V+i)*(*(V+j))*(*(*(Y+i)+j))*cos( *(delta+i)-(*(delta+j))-
(*(*(teta+i)+j;( ( ((
)* Qcal+i)+=( *(V+i)*(*(V+j))*(*(*(Y+i)+j))*sin( *(delta+i)-(*(delta+j))-
(*(*(teta+i)+j;(( ((
{
{

for(i=0;i<n;i(++
}
)*PG+i)=(*(Pcal+i))+(*(PL+i;((
)*QG+i)=(*(Qcal+i))+(*(QL+i;((
{
-------------------------------------------------------------------------------//

for(i=0;i<n;i(++
)*delta+i)*=(180/3.141592654 ;(

-------------------------------------------------------------------------------//
printf("\n\n\n;("
printf("\t\t\t\t repeat is:%d",o;(
printf("\n-------------------------------------------------------------------------------\n;("
printf("\tbus\tvol\t\tdeg\t\tPG\t\tQG;("
printf("\n-------------------------------------------------------------------------------\n;("
for(i=0;i<n;i(++
}
printf("\t%d\t%lf\t%lf\t%lf\t%lf",i+1,*(V+i),*(delta+i),*(PG+i),*(QG+i;( (
printf("\n\n;("
{
printf("\n-------------------------------------------------------------------------------\n;("

printf("\tbus\tPL\t\tQL;("
printf("\n-------------------------------------------------------------------------------\n;("
for(i=0;i<n;i(++
}
printf("\t%d\t%lf\t%lf",i+1,*(PL+i),*(QL+i;((
printf("\n\n;("
{
printf("\n-------------------------------------------------------------------------------\n;("

-------------------------------------------------------------------------------//
 for(i=0;i<n;i(++
}
  free(*(Y+i;((
  free(*(teta+i;((
{

 for(i=0;i<(n1+m);i(++
}
  free(*(J+i;((
```

```
{
 free(help1;(
 free(help2;(
 free(help3;(
 free(J;(
 free(V;(
 free(Y;(
 free(PL;(
 free(QL;(
 free(PG;(
 free(QG;(
 free(Pcal;(
 free(Qcal;(
 free(del_delta_V;(
 free(teta;(
 free(delta;(
 free(del_P_Q;(
 getch;()

 return 0;
//{ end of program.
**********************************************************************//
***********
**********************************************************************//
***********
void get_array_Y(double **Y,int n(
}
 for(i=0;i<n;i(++
}
  for(j=0;j<n;j(++
}
   printf("\n Please enter value for Y[%d][%d]:",i+1,j+1;(
   scanf("%lf", (*(Y+i)+j;( (
{
{
{
**********************************************************************//
***********
void get_array_teta(double **teta,int n(
}

double a;
 for(i=0;i<n;i(++
}
  for(j=0;j<n;j(++
}
   printf("\n Please enter value for Y bus degree[%d][%d]:",i+1,j+1;(
   scanf("%lf",&a;(
)*)*    teta+i)+j)=(a* 3.141592654)/180;
{
```

```
{
{

**********************************************************************//
**********
//resive input data.
void input_data(double *V,double *PL,double *QL,double *PG,int *choose,int n(
}
  for(i=0;i<n;i(++
}
  printf("\n Please choose one of the items below for bus(%d):\n",i+1;(
  printf("\n1)slack bus.\n;("
  printf("\n2)load bus.\n;("
  printf("\n3)control bus.\n;("
  printf("\n4)generator bus.\n;("
  printf("\n5)trans bus.\n;("
  scanf("%d",choose+i;(

  switch(*(choose+i((
}
  case 1:
    printf("\n Please enter value for voltag of slack bus;(":
    scanf("%lf",V+i;(
    printf("\n Please enter value for PL bus(%d):",i+1;(
    scanf("%lf",PL+i;(
    printf("\n Please enter value for QL bus(%d):",i+1;(
    scanf("%lf",QL+i;(
   break;

  case 2:
    printf("\n Please enter value for PL bus(%d):",i+1;(
    scanf("%lf",PL+i;(
    printf("\n Please enter value for QL bus(%d):",i+1;(
    scanf("%lf",QL+i;(
  break;

  case 3:
    printf("\n Please enter value for PL bus(%d):",i+1;(
    scanf("%lf",PL+i;(
    printf("\n Please enter value for QL bus(%d):",i+1;(
    scanf("%lf",QL+i;(
    printf("\n Please enter value for voltag of bus(%d):",i+1;(
    scanf("%lf",V+i;(
  break;

  case 4:
    printf("\n Please enter value for voltag of bus(%d):",i+1;(
    scanf("%lf",V+i;(
    printf("\n Please enter value for PG bus(%d):",i+1;(
    scanf("%lf",PG+i;(
```

```
    printf("\n Please enter value for PL bus(%d):",i+1;(
    scanf("%lf",PL+i;(
    printf("\n Please enter value for QL bus(%d):",i+1;(
    scanf("%lf",QL+i;(
  break;
//{   end of switch
//{  end of for

{
*****************************************************************//
**********
void inverse_matrix(double **J,int n1,int m(
}

int k,a,t,l=0;
double **B,**w,**mat,**inv,*p,*s,A,b;



 B=(double **)malloc((n1+m)*sizeof(double;((*
   for(i=0;i<(n1+m);i(++
)*  B+i)=(double *)malloc((n1+m)*sizeof(double;((

 w=(double **)malloc((n1+m)*sizeof(double;((*
   for(i=0;i<(n1+m);i(++
)*  w+i)=(double *)malloc((n1+m)*sizeof(double;((

 mat=(double **)malloc((n1+m)*sizeof(double;((*
   for(i=0;i<(n1+m);i(++
)*  mat+i )=(double *)malloc((n1+m)*sizeof(double;((


 inv=(double **)malloc((n1+m)*sizeof(double;((*
   for(i=0;i<(n1+m);i(++
)*  inv+i)=(double *)malloc((n1+m)*sizeof(double;((


  p=(double *)malloc((n1+m)*sizeof(double;((
  s=(double *)malloc((n1+m)*sizeof(double;((

  for(i=0;i<(n1+m);i(++
}
)*    s+i)=0;
)*    p+i)=0;
{

 for(i=0;i<(n1+m);i(++
}
  for(j=0;j<(n1+m);j(++
)*)*    B+i)+j)=(*(*(J+i)+j;((
```

```
{
----------------------------------------//
for(t=0;t<(n1+m);t(++
}

  for(i=0;i<(n1+m);i(++
}
   for(j=0;j<(n1+m);j(++
}
    if(i==j(
}
)*      s+l)+=*(*(B+i)+j;(
{
{
{
  l;++
-----------------------------------//

 for(k=0;k<(n1+m);k(++
}
  for(i=0;i<(n1+m);i(++
}
   for(j=0;j<(n1+m);j(++
}
)*)*    w+k)+i)+=(*(*(B+k )+j))*(*(*(J+j)+i;((
//{    end of for (j.(
//{   end of for(i.(
//{ end of for(k.(

----------------------------------//
 for(i=0;i<(n1+m);i(++
}
   for(j=0;j<(n1+m);j(++
}
)*)*    B+i)+j)=(*(*(w+i)+j;((
)*)*    w+i)+j)=0;
{
{

//{end of for(t.(
----------------------------------------//
 for(i=0,b=1;i<(n1+m);i++,b(++
}

  for(j=i,a=0;j>-1;j--,a(++
}
   if(j==i(
    A=1;
    else
    A=(*(p+(a-1;(((
```

```
)*)  p+i))+=(-1/b)*(A*(*(s+j;(((
{
{


  if( *(p+(n1+m-1))==0(
}
  printf("\n*(the inverse of matrix is impossible)*\n;("
  getch;()
//{  end of if.
-----------------------------------------------//

 for(i=0;i<(n1+m);i(++
}
   for(j=0;j<(n1+m);j(++
}
)*)*    inv+i)+j)=0;
)*)*    mat+i )+j)=*(*(J+i)+j;(
)*)*    B+i)+j)=*(*(J+i)+j;(
{
{
-----------------------------------------------//
  for(t=0,a=(n1+m-3);t<(n1+m-1);t++,a(--
}
   if(t==(n1+m-2( (
   A=1;
   else
   A=*(p+a;(

 for(i=0;i<(n1+m);i(++
}
   for(j=0;j<(n1+m);j(++
}
)*)*    mat+i )+j)*=A;
)*)*    inv+i)+j)+=*(*(mat+i)+j;(
{
{

  if(t==(n1+m-2( (
  break;


 for(k=0;k<(n1+m);k(++
}
  for(i=0;i<(n1+m);i(++
}
   for(j=0;j<(n1+m);j(++
}
)*)*   w+k)+i)+=(*(*B+k )+j))*(*(*(J+j)+i;((
```

```
//{   end of for (j.(
//{  end of for(i.(
//{ end of for(k.(


  for(i=0;i<(n1+m);i(++
}
   for(j=0;j<(n1+m);j(++
}
)*)*   B+i)+j)=*(*(w+i)+j;(
)*)*   mat+i )+j)=*(*(w+i)+j;(
)*)*   w+i)+j)=0;
{
{

//{ end of for (t.(

----------------------------------------//
  for(i=0;i<(n1+m);i(++
}
   for(j=0;j<(n1+m);j(++
}
    if(i!=j(
)*)*    mat+i )+j)=0;

    else if(i==j(
)*)*    mat+i )+j)=*(p+(n1+m-2;((
{
{

 for(i=0;i<(n1+m);i(++
}
   for(j=0;j<(n1+m);j(++
}
)*)*   inv+i)+j)+=*(*(mat+i)+j;(
)*)*   inv+i)+j)*=(-1/(*(p+(n1+m-1;( (((
)*)*   J+i)+j)=*(*(inv+i)+j;(
{
{
----------------------------------------------//
 for(i=0;i<(n1+m);i(++
 free(*(B+i;((
 free(B;(


 for(i=0;i<(n1+m);i(++
 free(*(mat+i ;((
 free(mat;(
```

```
 for(i=0;i<(n1+m);i(++
 free(*(inv+i;((
 free(inv;(

 for(i=0;i<(n1+m);i(++
 free(*(w+i;((
 free(w;(

 free(s;(
 free(p;(


{

**********************************************************************//
************************************************************
 void full_Pcal(double *Pcal,double *V,double *delta,double **Y,double **teta,int
n,int o,double *help(
}

 for(i=1;i<n;i(++
}
   for(j=0;j<n;j(++
)*Pcal+i)+=(*(V+i))*(*(V+j))*(*(*(Y+i)+j))*cos( *(delta+i)-(*(delta+j))-
(*(*(teta+i)+j;( ((
{

 if(o==0(
}
 for(i=0;i<n;i(++
)*  help+i)=(*(Pcal+i;((
{

 else if(o>0(
}
  for(i=0;i<n;i(++
}
)*  Pcal+i)-=(*(help+i;((
)*  help+i)=*(Pcal+i;(
{
//{ end of else.
{
**********************************************************************//
************************************************************
 void full_Qcal(double *Qcal,double *V,double *delta,double **Y,double **teta,int
n,int *find,int o,double *help(
}
 int l;

 for(i=1,l=0;i<n;i(++
```

```
}
 if( *(find+l)==i(
}
  for(j=0;j<n;j(++
)*Qcal+i)+=(*(V+i))*(*(V+j))*(*(*(Y+i)+j))*sin(*(delta+i)-(*(delta+j))-
(*(*(teta+i)+j;( ((
  l; ++
//{  end of if.
//{ end of for.

 if(o==0(
}
 for(i=0;i<n;i(++
)*  help+i)=(*(Qcal+i;((
{
 else if(o>0(
}
  for(i=0;i<n;i(++
}
)*  Qcal+i)-=(*(help+i;((
)*  help+i)=(*(Qcal+i;((
{
//{ end of else.
{
**********************************************************************//
**********************************************************************
 void full_del_P_Q(double *del_P_Q,double *PG,double *PL,double *Pcal,double
*QG,double *QL,double *Qcal,int *find,int n,int m(
}
 int n1,l;
 n1=n-1;
 for(i=0;i<n1;i(++
)* del_P_Q+i)=(*(PG+(i+1)))-(*(PL+(i+1)))-(*(Pcal+(i+1;(((

 for(i=n1,l=0;i<(n1+m);i++,l(++
)* del_P_Q+i)=(*(QG+(*(find+l))))-(*(QL+(*(find+l))))-(*(Qcal+(*(find+l;((((
{
**********************************************************************//
**********************************************************************
***
 void full_del_delta_V(double *del_delta_V,double *del_P_Q,double *delta,double
*V,int *find,double **J,int n1,int m,int o,double *help(
}
  int l;

 for(i=0;i<(n1+m);i(++
}
  for(j=0;j<(n1+m);j(++

)*  del_delta_V+i)+=(*(*(J+i )+j))*(*(del_P_Q+j;((
```

```
{

 if(o==0(
}
 for(i=0;i<(n1+m);i(++
)*  help+i)=(*(del_delta_V+i;((
{

 else if(o>0(
}
  for(i=0;i<(n1+m);i(++
}
)*  del_delta_V+i)-=(*(help+i;((
)*  help+i)=(*(del_delta_V+i;((
{
//{ end of else.

 for(i=0;i<n1;i(++
}
)* delta+(i+1))=( (*(delta+(i+1)))+(*(del_delta_V+i;( ((
{

 for(i=n1,l=0;i<(n1+m);i++,l(++
}
)* V+(*(find+l)))=( (*(V+(*(find+l))))+(*(del_delta_V+i;( ((
{
{
**********************************************************************//
***********
double test(double *del_P_Q,int n1,int m(
}
 double a;

 a=-1;
 for(i=0;i<(n1+m);i(++
}
  if( fabs(*(del_P_Q+i)) > a(
  a=fabs(*(del_P_Q+i;((
//{ end of for.
 return (a;(
{
**********************************************************************//
***********
```