مجله علمی – پژوهشی
مهندسی عمران مدرس
دوره هفدهم، شماره ۱، سال ۱۳۹۶

یادداشت تحقیقاتی:

پیاده‌سازی بهینه روش بدون شبکه توابع پایه نمایی روی بسترهای مختلف نرم‌افزاری و مقایسه کارایی آنها

فرشید مسیبی[1*]

۱- استادیار دانشکده مهندسی عمران و حمل و نقل، دانشگاه اصفهان

[*]**mossaiby@eng.ui.ac.ir**

تاریخ دریافت: [۹٤/۱۰/۰۲] تاریخ پذیرش: [۹۵/۰۳/۱۹]

چکیده- در دهه‌های اخیر روش‌های بدون شبکه مورد توجه پژوهشگران قرار گرفته‌اند. هزینه بالای تولید شبکه المان‌بندی، چه در بعد محاسباتی و چه در بعد نیروی انسانی متخصص یکی از مهم‌ترین دلایل این امر به شمار می‌رود. روش توابع پایه نمایی یکی از این روش‌ها است که در چند سال اخیر برای حل انواع معادلات دیفرانسیل پاره‌ای در مسائل مختلف علوم مهندسی با موفقیت به کار رفته است. در این مقاله پیاده‌سازی این روش روی بسترهای مختلف نرم‌افزاری بحث شده و کارایی نسبی آنها با یکدیگر مقایسه می‌شود. نتایج نشان می‌دهد با پیاده‌سازی مناسب می‌توان خطای ناشی از حل عددی را به شدت کاهش داد. کارایی نسبی انجام حل با استفاده از زبان‌های برنامه‌نویسی معمول مانند C++ در مقایسه با بسته‌های نرم‌افزاری ریاضی همواره یکی از سوالات رایج هنگام استفاده از این بسته‌های نرم‌افزاری است. در این پژوهش نشان داده می‌شود در صورت پیاده‌سازی بهینه روش توابع پایه نمایی این نسبت بین ۲/۵ تا ٦ متغیر است.

واژگان کلیدی: **روش توابع پایه نمایی، شبه وارون، جداسازی مقدار تکین، معادلات دیفرانسیل پاره‌ای، پیاده‌سازی بهینه.**

۱- مقدمه

روش المان محدود یکی از روش‌های حل عددی معادلات دیفرانسیل است که در دهه‌های اخیر توسعه فروانی پیدا کرده و کاربردهای زیادی یافته است. این روش محبوبیت خود را مدیون عواملی مانند سادگی نسبی پیاده‌سازی رایانه‌ای، کارایی بالا و نیز امکان استفاده در گستره بسیار وسیعی از مسائل است. با این وجود هنوز مسائل زیادی وجود دارند که روش المان محدود قادر به حل آنها نبوده و یا در حل آنها دچار مشکلات اساسی است. نمونه‌ای از این مسائل را می‌توان در مرجع [1] جستجو نمود. یکی از پرهزینه‌ترین قسمت‌های استفاده از این روش ساخت شبکه المان‌بندی است. پیچیدگی ساخت این شبکه‌ها به علت نیاز روش المان محدود به شکستن انتگرال‌های دامنه به روی زیردامنه‌ها است. این زیردامنه‌ها که در این روش المان نامیده می‌شوند باید به گونه‌ای ساخته شوند که جز مرز خود با سایر المان‌ها اشتراکی نداشته و اجتماع آنها نیز برابر دامنه اصلی باشد. این شکل تقسیم دامنه به زیردامنه‌ها که اصطلاحاً افراز[1] نامیده می‌شود در حالت کلی و به ویژه در مسائل سه‌بعدی بسیار پرهزینه است. این هزینه تنها در بعد محاسباتی نبوده و شامل نیاز به نیروی متخصص انسانی نیز می‌شود. از این روست که در سال‌های اخیر روش‌های بدون شبکه مورد توجه پژوهشگران قرار گرفته است. یکی از روش‌های بدون شبکه که به تازگی در مسائل زیادی با موفقیت مورد استفاده قرار گرفته است، روش توابع پایه نمایی[2] است.

1. partition
2. exponential basis functions method

این روش ابتدا بـه وسـیله‌ی برومنـد و همکـاران در سـال ۲۰۰۹ ابداع شد [2] و تـاکنون بـرای حـل مسـائل الاستیسـیته، انتشار موج الاسـتیک، خمـش صـفحات، دینامیـک سـیالات و بسیاری مسائل دیگر به کار رفته است [3–15]. سادگی روابط و پیاده‌سازی رایانه‌ای، امکان توسعه سریع به انواع معادلات خطی با ضرائب ثابت، عدم نیاز به انتگرال‌گیری و گسسته‌سازی دامنه حل از دیگر مزایای آن به شمار می‌رود.

پیاده‌سازی مناسب می‌تواند تاثیر بسیار زیـادی بـر کـارایی این روش در حل مسائل داشـته باشـد. سـرعت اجـرا، کـاهش خطاهای عددی ناشی از گرد کردن[1]، وجـود توابـع کتابخانـه‌ای مناسب و سهولت توسعه برنامه از مواردی است که بایـد بـرای یک پیاده‌سازی مناسب در نظر گرفته شود. معمولاً توسـعه یـک روش عددی در مراحل اولیه روی بسته‌های نرم‌افزاری ریاضـی مانند متمتیکا[2] و متلب[3] انجام می‌گیرد. کـارایی ایـن بسـته‌هـای نرم‌افزاری در اجرای توابع کتابخانه‌ای خود در مقایسه با برنامـه نوشته‌شده به وسیله‌ی کاربر بسیار متفاوت است. ایـن موضـوع باعث می‌شود مقایسه روش‌های تازه توسعه یافته با روش‌هـای عددی قدیمی‌تر که به وسیله‌ی زبان‌هـای برنامـه‌نویسـی عـادی مانند C++ و فرترن[4] نوشته شده‌اند، بسیار مشکل باشد و نقـاط ضعف روش‌های جدیـد بـه سـرعت مشـخص نشـود. در ایـن پژوهش برای نزدیک شدن بـه پاسـخی بـرای ایـن مشـکلات، کارایی روش توابع پایه نمایی روی بسته نرم‌افـزاری متمتیکـا و پیاده‌سازی آن به وسیله‌ی زبان برنامه‌نویسی C++ مورد مقایسـه قرار می‌گیرد. در هر دو مورد روش‌های مختلف پیـاده‌سـازی و کارایی نسبی قسمت‌های مختلف حل بررسی می‌شود.

۲- مروری بر روش توابع پایه نمایی

روش توابع پایه نمایی یک روش بدون شبکه مبتنـی بـر مرز[5] است. در این روش برای تقریـب متغیرهـای میـدان از یک ترکیب خطی از توابع پایـه نمـایی بـا ضـرائب مجهـول استفاده می‌شود. این توابع به شکلی انتخـاب مـی‌شـوند کـه معادله دیفرانسیل همگن را بدون توجه به شرایط مـرزی آن برآورده نمایند. از سوی دیگـر ضـرائب مجهـول بـه شـکلی بدست می‌آیند که شرایط مرزی به صورت تقریبـی بـرآورده شود. یک مساله معادله دیفرانسـیل شـامل معادلـه حـاکم[6] و شرایط مرزی در حالت کلی به صورت زیـر در نظـر گرفتـه می‌شود:

$$\begin{cases} \mathbf{L}\mathbf{u} = \mathbf{f} & \text{in } \Omega \\ \mathbf{L}_D\mathbf{u} = \mathbf{f}_D & \text{on } \Gamma_D \\ \mathbf{L}_N\mathbf{u} = \mathbf{f}_N & \text{on } \Gamma_N \end{cases}, \quad \begin{matrix} \Gamma_D \cup \Gamma_N = \Gamma = \partial\Omega \\ \Gamma_D \cap \Gamma_N = \varnothing \end{matrix} \qquad (۱)$$

که در آن $\mathbf{u}$ تابع مجهول، $\mathbf{L}$ عملگر دیفرانسیلی خطـی با ضرایب ثابت، $\mathbf{L}_D$ و $\mathbf{L}_N$ عملگرهای دیفرانسـیلی مـرزی روی مرزهای ضروری[7] و طبیعی[8] بـوده و $\mathbf{f}$، $\mathbf{f}_D$ و $\mathbf{f}_N$ بـه ترتیب توابع سمت راست مناسب در معادلات هسـتند. Ω و Γ دامنـه حـل و مـرز آن بـوده و Γ_D و Γ_N بخـش‌هـای ضروری و طبیعی این مرز را نشان می‌دهد. از آنجا که تاکید این پژوهش بر پیاده‌سازی بهینه این روش بوده و روش حل در شکل همگن و غیرهمگن بسیار به یکدیگر شباهت دارد، تنها شکل همگن معادله حاکم ($\mathbf{f} = \mathbf{0}$) در حالـت دوبعـدی در نظر گرفته شده است. شکل کلی این روش در مرجع [۲] آمده است. برای حل این مساله معادله دیفرانسیل بـا کمـک روش توابع پایه نمایی، تقریبی از متغیرهای میدان به صورت زیر در نظر گرفته می‌شود:

$$\mathbf{u} \approx \hat{\mathbf{u}} = \sum_{k=1}^{m} c_k \boldsymbol{\psi}_k \qquad (۲)$$

در رابطـه بـالا، c_k ضـرائب مجهـول و $\boldsymbol{\psi}_k$ توابـع پایـه نمایی هستند. توابع پایه در این روش به شکل زیـر تعریـف می‌شوند:

$$\boldsymbol{\psi}_k = \boldsymbol{\Lambda}_k \exp(\alpha_k x + \beta_k y) \qquad (۳)$$

که در آن α_k و β_k اعدادی مختـلط و $\boldsymbol{\Lambda}_k$ یـک بـردار وابسته به α_k و β_k است. از آنجایی که در این روش توابع پایه باید در معادله حاکم صدق نماید:

$$\mathbf{L}\boldsymbol{\psi}_k = \mathbf{0} \qquad (٤)$$

آشکار است در این حالت با توجه بـه خطـی بـودن $\mathbf{L}$

1 round-off error
2 Mathematica
3 Matlab
4 Fortran
5 boundary method

6 governing differential equation
7 essential
8 natural

رابطه زیر همواره برقرار خواهد بود:

$$\mathbf{L}\hat{\mathbf{u}} = \mathbf{L}\sum_{k=1}^{m} c_k \boldsymbol{\psi}_k = \sum_{k=1}^{m} c_k \mathbf{L}\boldsymbol{\psi}_k = \mathbf{0} \qquad (۵)$$

به عبارت دیگر معادله حاکم به صورت دقیق برآورده خواهد شد و تنها نیاز است شرایط مرزی برآورده شود. برای این کار مرز دامنه حل Γ به وسیله‌ی یک سری از نقاط گسسته‌سازی می‌شود. اگر اپراتور دیفرانسیلی $\mathbf{L}_B$ و تابع سمت راست آن روی مرز به شکل:

$$\mathbf{L}_B = \begin{cases} \mathbf{L}_D & \mathbf{x}_j \in \Gamma_D \\ \mathbf{L}_N & \mathbf{x}_j \in \Gamma_N \end{cases}, \quad \mathbf{f}_B = \begin{cases} \mathbf{f}_D & \mathbf{x}_j \in \Gamma_D \\ \mathbf{f}_N & \mathbf{x}_j \in \Gamma_N \end{cases} \qquad (۶)$$

تعریف شود که در آن $\mathbf{x}_j$ مختصات یک نقطه مرزی است، رابطه زیر به دست می‌آید:

$$\mathbf{L}_B\hat{\mathbf{u}} = \mathbf{L}_B\sum_{k=1}^{m} c_k \boldsymbol{\psi}_k = \sum_{k=1}^{m} c_k \mathbf{L}_B\boldsymbol{\psi}_k = \mathbf{f}_B \qquad (۷)$$

با برآورد رابطه بالا در محل نقاط مرزی و ترکیب معادلات به صورت ستونی، دستگاه معادلات خطی زیر بدست می‌آید:

$$\sum_{k=1}^{m} c_k \mathbf{v}_k = \mathbf{h} \qquad (۸)$$

که در آن:

$$(\mathbf{v}_k)_j = (\mathbf{L}_B\boldsymbol{\psi}_k)\big|_{\mathbf{x}_j}, \quad (\mathbf{h})_j = \mathbf{f}_B\big|_{\mathbf{x}_j}, \quad j = 1,\cdots,n \qquad (۹)$$

در رابطه بالا n تعداد نقاط مرزی است. دستگاه معادلات فوق را می‌توان به صورت زیر بازنویسی نمود

$$\mathbf{v}\mathbf{c} = \mathbf{h} \qquad (۱۰)$$

که در آن:

$$\mathbf{c} = \langle c_1 \quad c_2 \quad \cdots \quad c_m \rangle^T, \quad \mathbf{v} = \left[\mathbf{v}_1^T \quad \mathbf{v}_2^T \quad \cdots \quad \mathbf{v}_m^T\right]^T \qquad (۱۱)$$

از آنجا که توابع پایه به کار رفته لزوماً بر یکدیگر عمود نبوده و تعداد آنها (m) نیز در حالت کلی با تعداد نقاط مرزی (n) برابر نیست، $\mathbf{v}$ یک ماتریس مستطیلی با مرتبه ناکامل[1] خواهد بود. دستگاه معادلات خطی (۱۰) در این حالت جواب یکتا ندارد. چنانچه از روش حداقل مربعات خطا استفاده شود، بهترین جواب این دستگاه معادلات به صورت زیر بدست می‌آید:

$$\left\|\mathbf{v}\mathbf{c} - \mathbf{h}\right\|_2 = \min \rightarrow \mathbf{c} = \mathbf{v}^{+}\mathbf{h} \qquad (۱۲)$$

که در آن $\mathbf{v}^{+}$ وارون تعمیم یافته[2] مور–پنروز[3] است.

برای برقراری رابطه (٤)، باید رابطه‌ای میان α_k و β_k و نیز $\boldsymbol{\Lambda}_k$ برقرار باشد. بدین منظور اگر رابطه (۳) در رابطه (٤) قرار داده شود، با توجه به این که $\mathbf{L}$ یک عملگر دیفرانسیلی با ضرائب ثابت است:

$$\begin{aligned} \mathbf{L}\boldsymbol{\psi}_k &= \mathbf{L}[\boldsymbol{\Lambda}_k \exp(\alpha_k x + \beta_k y)] \\ &= \mathbf{Q}(\alpha_k,\beta_k)\boldsymbol{\Lambda}_k \exp(\alpha_k x + \beta_k y) = \mathbf{0} \end{aligned} \qquad (۱۳)$$

از آنجا که تابع نمایی همواره مخالف صفر است، برای برقراری رابطه (۱۳) به شکل غیربدیهی ($\boldsymbol{\Lambda}_k \neq \mathbf{0}$) باید

$$\det \mathbf{Q}(\alpha_k,\beta_k) = 0, \quad \boldsymbol{\Lambda}_k \in \operatorname{null}\mathbf{Q}(\alpha_k,\beta_k) \qquad (۱٤)$$

که در آن $\operatorname{null}\mathbf{Q}(\alpha_k,\beta_k)$ فضای پوچ ماتریس $\mathbf{Q}(\alpha_k,\beta_k)$ است. فضای پوچ یک ماتریس به صورت زیر تعریف می‌شود:

$$\operatorname{null}\mathbf{A} = \{\mathbf{v} \mid \mathbf{A}\mathbf{v} = \mathbf{0}\} \qquad (۱۵)$$

معادله اول رابطه (۱٤) معادله مشخصه[4] نامیده می‌شود. این معادله در حالت دوبعدی یک رابطه میان α_k و β_k بدست می‌دهد و با انتخاب یکی از این مقادیر، می‌توان مقدار دیگر را محاسبه نمود. در هر صورت اگر رابطه (۱٤) برقرار باشد، معادله دیفرانسیل حاکم به صورت دقیق برقرار خواهد شد. برای توضیحات بیشتر در مورد چگونگی انتخاب این اعداد به مرجع [2] مراجعه نمایید.

استفاده از اعداد مختلط همواره مورد پسند کاربران نیست. به سادگی می‌توان نشان داد که با استفاده از رابطه معروف اویلر، می‌توان محاسبات را تنها به کمک اعداد حقیقی انجام داد. با تجزیه توابع شکل به قسمت‌های حقیقی و موهومی، توابع پایه جدیدی به دست می‌آید که معادله حاکم را برآورده نموده و می‌تواند به جای توابع پیشنهادی قبلی مورد استفاده قرار گیرد. در این روند تعداد توابع پایه و در نتیجه تعداد ضرائب مجهول دو برابر می‌شود، اما از آنجا که برای ذخیره نمودن هر عدد مختلط حافظه‌ای دو برابر مقدار لازم برای ذخیره نمودن یک عدد حقیقی مورد نیاز است، حافظه کلی مورد استفاده تغییری نمی‌کند. از آنجا که

1 rank deficient

2 generalized inverse
3 Moore-Penrose
4 characteristic equation

برخی از کتابخانه‌های توابع از اعداد مختلط پشتیبانی نمی‌کنند، در این پژوهش از این شکل روش برای حل مسائل استفاده شده است.

۳- پیاده‌سازی روش توابع پایه نمایی

نگاهی به قسمت قبل نشان می‌دهد با ساخت ماتریس $\mathbf{v}$ که در حقیقت محاسبه توابع پایه ψ_k در محل نقاط $\mathbf{x}_j$ است و محاسبه ضرائب $\mathbf{c}$، عملاً محاسبه $\hat{\mathbf{u}}$ در هر نقطه از دامنه بسیار ساده است. تنها قسمتی که از نظر محاسباتی و پیاده‌سازی کمی پیچیده به نظر می‌رسد، محاسبه وارون تعمیم‌یافته ماتریس $\mathbf{v}$ است. محاسبه این وارون در بسته‌های نرم‌افزاری ریاضی از طریق توابع کتابخانه‌ای امکان‌پذیر است. این وارون به علت خواص منحصر بفرد آن کاربردهای بسیار زیادی در بسیاری از شاخه‌های علوم دارد [16]. برای محاسبه این وارون روش‌های بسیار زیادی توسعه داده شده است. بسیاری از این روش‌ها در محدوده‌ای از مسائل کاربرد داشته و در حالت کلی نمی‌توانند وارون تعمیم‌یافته هر ماتریس دلخواه را محاسبه نمایند. برای بررسی کارایی، برخی از این روش‌ها به وسیله‌ی نویسنده به صورت کامل پیاده‌سازی شد. نتایج بدست آمده همراه با توصیه‌های موجود در بیشتر مراجع مانند مراجع [16] نشان داد که به علت پایین بودن مرتبه ماتریس $\mathbf{v}$ و ابعاد نسبتاً بزرگ آن، تنها روش مطمئن برای محاسبه وارون تعمیم‌یافته یک ماتریس، استفاده از جداسازی مقدار تکین[1] است. در این روش یک ماتریس دلخواه مانند $\mathbf{A}$ به صورت زیر جداسازی می‌شود:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H \qquad (۱۶)$$

که در آن $\mathbf{U}$ و $\mathbf{V}$ به ترتیب ماتریس شامل بردارهای تکین چپ و راست بوده و بالانویس H ترانهاده مزدوج مختلط[2] یک ماتریس را نشان می‌دهد. همچنین $\mathbf{\Sigma}$ یک قطری است که مقادیر تکین را روی قطر اصلی خود جا داده است. این مقادیر همواره حقیقی و بزرگتر یا مساوی صفر هستند. یک ماتریس مربعی با مقادیر تکین بزرگتر از صفر دارای وارون بوده و وارون آن با وارون تعمیم‌یافته برابر است.

با موجود بودن این جداسازی از یک ماتریس، وارون تعمیم‌یافته آن به صورت زیر تعریف می‌شود:

$$\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^H \qquad (۱۷)$$

که در آن ماتریس قطری $\mathbf{\Sigma}^+$ به صورت زیر تعریف می‌شود:

$$(\mathbf{\Sigma}^+)_{ii} = \begin{cases} (\mathbf{\Sigma})_{ii}^{-1} & (\mathbf{\Sigma})_{ii} \neq 0 \\ 0 & (\mathbf{\Sigma})_{ii} = 0 \end{cases} \quad (\text{no sum on } i) \qquad (۱۸)$$

در هنگام پیاده‌سازی عددی، به جای مقایسه مقادیر تکین با صفر، از یک عدد کوچک مانند ε استفاده می‌کنند که معمولاً به صورت زیر تعریف می‌شود:

$$\varepsilon = 100\varepsilon_m \max \mathbf{\Sigma} \qquad (۱۹)$$

که در آن $\max \mathbf{\Sigma}$ بزرگترین مقدار تکین ماتریس بوده و ε_m حد بالای خطای نسبی ناشی از گرد کردن اعداد در رایانه مورد استفاده (ε ماشین[3]) است. برای متغیرهای با دقت مضاعف[4] این مقدار برابر $۲/۲۲۰۴۴۶۰۴۹۲۵۰۳۱۳\times۱۰^{-۱۶}$ است. در پیاده‌سازی C++ برای محاسبه جداسازی تکین از تابع DGESDD متعلق به کتابخانه توابع LAPACK[5] [17] استفاده شده است.

در بسته نرم‌افزاری متمتیکا دستور PseudoInverse برای محاسبه وارون تعمیم‌یافته یک ماتریس به کار می‌رود. مطابق مستندات فنی، این دستور نیز از جداسازی مقدار تکین برای این کار بهره می‌گیرد. همچنین از دستور SingularValueDecomposition می‌توان برای محاسبه جداسازی مقدار تکین استفاده نمود. رابطه (17) شامل ضرب سه ماتریس در یکدیگر می‌باشد. این ضرب می‌تواند به صورت یکجا و یا در دو مرحله انجام شود. در هر دو صورت حجم محاسبات بسیار بالا است. یک روش جهت کاهش محاسبات استفاده از قطری بودن ماتریس $\mathbf{\Sigma}^+$ و مقیاس نمودن سطرها و یا ستون‌های ماتریس‌های $\mathbf{U}$ و $\mathbf{V}$ به صورت مناسب است. در پیاده‌سازی انجام شده با C++ مشاهده شد انجام این کار نه تنها حجم محاسبات را کاهش می‌دهد، بلکه می‌تواند در کاهش خطاهای عددی نیز بسیار موثر باشد. بهینه‌سازی دیگری که در روند پیاده‌سازی انجام گرفت کاهش مجدد حجم محاسبات و

1 singular value decomposition
2 complex conjugate transpose

3 machine epsilon
4 double precision
5 linear algebra package

خطاهای عددی با استفاده از روابط (۱۲ و ۱۷) است. از رابطه (۱۷) واضح است که در نهایت حاصل‌ضرب ماتریس وارون تعمیم‌یافته در یک بردار مورد احتیاج است. چنانچه از رابطه (۱۷) این وارون به صورت زیر نوشته شود

$$\mathbf{v}^{+} = \mathbf{V}\boldsymbol{\Sigma}^{+}\mathbf{U}^{H} \qquad (۲۰)$$

با جایگذاری آن در رابطه (۱۲)

$$\mathbf{c} = \mathbf{v}^{+}\mathbf{h} = \mathbf{V}\boldsymbol{\Sigma}^{+}\mathbf{U}^{H}\mathbf{h} = \mathbf{V}\left[\boldsymbol{\Sigma}^{+}(\mathbf{U}^{H}\mathbf{h})\right] \qquad (۲۱)$$

با توجه به خاصیت شرکت‌پذیری ضرب ماتریس‌ها، در رابطه بالا تنها ضرب ماتریس در بردار مورد نیاز است که حجم بسیار کمتری از عملیات را در پی داشته و منجر به خطای گردکردن بسیار کمتری می‌شود. نتایج نشان می‌دهد کاهش خطا در اثر پیاده‌سازی این مورد در بسته ریاضی متمتیکا هم به همان اندازه قابل توجه و موثر است.

بعد از پژوهشهای انجام شده مشخص شد در کتابخانه توابع LAPACK که برای محاسبه جداسازی مقدار تکین به کار رفته است، توابع دیگری وجود دارد که مستقیماً می‌تواند مساله حداقل مربعات خطای مطرح شده در رابطه (۱۲) را با دادن مقادیر $\mathbf{v}$ و $\mathbf{h}$ بدون هیچ پیش شرطی روی مرتبه ماتریس $\mathbf{v}$ حل کند. در این روش بردار $\mathbf{c}$ بدون نیاز به محاسبه صریح ماتریس‌های $\mathbf{U}$، $\boldsymbol{\Sigma}$ و $\mathbf{V}$ بدست می‌آید و با توجه به عدم نیاز به استفاده از ضرب ماتریس در بردار مطرح شده در رابطه (۲۱) بسیار مقرون به صرفه است. سریع‌ترین تابعی که این کار را انجام می‌دهد تابع DGELSD است که از روش تقسیم و غلبه[1] استفاده می‌کند [17]. از این تابع برای محاسبه بردار $\mathbf{c}$ در پیاده‌سازی C++ استفاده شد و با توجه به کاهش بیشتر محاسبات، سرعت محاسبات افزایش یافته و خطاهای عددی نیز کاهش چشمگیری پیدا نمود. توضیحات بیشتر در مورد کتابخانه‌های توابع استفاده شده در بخش آتی ارائه خواهد شد.

در پیاده‌سازی متمتیکا نیز با جستجو در مستندات فنی مشخص شد تابع LinearSolve نیز هنگامی که مرتبه ماتریس کامل نباشد برای حل دستگاه معادلات از روش مشابهی استفاده می‌کند. با استفاده از این دستور خطاهای عددی مانند پیاده‌سازی C++ کاهش چشمگیری از خود نشان داد.

قسمت دیگری از روند حل که هر چند ساده است می‌تواند مورد بهینه‌سازی واقع شود، مرحله ساخت ماتریس $\mathbf{v}$ است. بهینه‌سازی که در این مورد می‌تواند انجام گیرد، ساخت این ماتریس به صورت موازی[2] در پیاده‌سازی C++ است که با استفاده از OpenMP[3] انجام شده و از تمام هسته‌های پردازنده برای ساخت ماتریس $\mathbf{v}$ به صورت همزمان استفاده می‌نماید. این روش برای ساخت ماتریس $\mathbf{v}$ هر چند ساده است، باید با دقت زیادی انجام شود تا از مشکلاتی مانند شرایط مسابقه[4] میان رشته‌های[5] برنامه جلوگیری شود.

در پیاده‌سازی متمتیکا نیز بهینه‌سازی قابل توجهی در ساخت ماتریس $\mathbf{v}$ با استفاده دستور Compile قابل انجام است. این دستور برای کامپایل کردن کد کاربر به نوعی کد میانی[6] برای افزایش سرعت اجرای آن به کار می‌رود. از محدودیت‌های این دستور می‌توان به عدم امکان استفاده از محاسبات نمادین[7] در کد کامپایل شده اشاره کرد. همچنین این دستور هیچ تاثیری بر سرعت اجرای توابع کتابخانه‌ای (مانند LinearSolve) ندارد. در نسخه‌های اخیر متمتیکا، امکان استفاده از پردازش موازی فراهم شده است. در صورت استفاده مناسب، این قابلیت باعث کاهش زمان اجرای برنامه با استفاده از تمام هسته‌های پردازنده می‌شود که مشابه استفاده از OpenMP در پیاده‌سازی C++ است. در بهینه‌سازی ساخت ماتریس ضرایب می‌توان از دستور ParallelTable که نسخه موازی دستور Table است، بهره برد. در بخش مربوط به نتایج عددی آثار هر کدام از این بهینه‌سازی‌ها بر سرعت اجرا و خطاهای عددی بررسی خواهد شد.

٤- کتابخانه‌های توابع استفاده شده در پیاده‌سازی C++

در قسمت قبل اشاره شد که کتابخانه توابع LAPACK در این پژوهش برای جداسازی مقدار تکین (۲۰) و حل دستگاه

1 divide and conquer

2 parallel
3 open multi-processing
4 race condition
5 threads
6 intermediate code
7 symbolic computations

معادلات خطی (۱۰) استفاده شده است. این کتابخانه توابع اولین بار در سال ۱۹۹۲ و به وسیله زبان فرترن توسعه داده شد و به سرعت تبدیل به یکی از کتابخانه‌های توابع استاندارد در کارهای عددی تبدیل گشت. حل دستگاه‌های معادلات خطی، مسائل حداقل مربعات خطا و نیز مسائل مقادیر ویژه و تکین از جمله کاربردهای این کتابخانه توابع است. این کتابخانه توابع خود از کتابخانه توابع سطح پایین‌تری به نام BLAS[1] [18] برای انجام محاسبات مربوط به بردارها و ماتریس‌ها استفاده می‌کند. کتابخانه توابع BLAS نیز یک کتابخانه توابع استاندارد کارهای عددی است که در سال ۱۹۷۹ به وسیله زبان فرترن توسعه داده شد. هر دو این کتابخانه‌ها به صورت رایگان/کد باز[2] در اختیار عموم قرار دارند و این موضوع باعث توسعه بسیار زیاد آنها شده است. به علت استاندارد بودن نام توابع و پارامترهای به کار رفته در آنها و نیز کاربرد بسیار زیاد این توابع، پیاده‌سازی‌های متفاوتی از آنها توسعه داده شده است. این پیاده‌سازی‌ها برای بسترهای نرم‌افزاری و سخت‌افزاری ویژه‌ای بهینه‌سازی شده‌اند. از جمله معروف‌ترین پیاده‌سازی‌های این کتابخانه‌ها می‌توان به کتابخانه‌های کد بسته[3] AMD ACML[4] [۱۹] و Intel MKL[5] [20] اشاره نمود که هر کدام برای پردازنده‌های ساخت این شرکت‌ها بهینه‌سازی شده‌اند. از آنجا که تقریباً تمام محاسبات زمان‌گیر در کتابخانه LAPACK به وسیله‌ی توابع BLAS انجام می‌شود، با استفاده از یک پیاده‌سازی بهینه BLAS می‌توان حتی با کمک پیاده‌سازی استاندارد LAPACK نیز به کارایی بسیار بالایی دست پیدا کرد. از جمله پیاده‌سازی‌های بسیار کارآمد BLAS می‌توان به ATLAS[6] [21] و OpenBLAS[7] [22] که هر دو پیاده‌سازی‌های رایگان/کد باز هستند، اشاره نمود. این روش در بخش آتی برای دستیابی به کارایی بالا در محاسبات مورد بررسی و مقایسه قرار خواهد گرفت.

۵- نتایج عددی

از آنجا که بهینه‌سازی‌های معرفی شده در قسمت قبل کاملاً کلی است، برای سادگی در ارائه مطالب از معادله لاپلاس که یک معادله اسکالر است، به عنوان نمونه استفاده می‌شود. این معادله که کاربرد زیادی در بسیاری از شاخه‌های علوم دارد، به صورت زیر بیان می‌شود:

$$Lu = \nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \qquad (۲۲)$$

با استفاده از رابطه فوق و نیز رابطه (۱۳) معادله مشخصه و مقدار Λ_k به صورت زیر بدست می‌آید:

$$Q(\alpha_k, \beta_k) = \alpha_k^2 + \beta_k^2 = 0, \quad \Lambda_k = 1 \qquad (۲۳)$$

با حل معادله بالا بر حسب α_k و β_k به صورت جداگانه و جایگذاری در رابطه (۳) و نامگذاری مناسب بر حسب یک متغیر مانند λ_k، توابع پایه به صورت زیر بدست می‌آید:

$$\exp(\lambda_k x \pm \mathrm{i}\lambda_k y), \exp(\pm \mathrm{i}\lambda_k x + \lambda_k y) \qquad (۲۴)$$

که در آن $\mathrm{i} = \sqrt{-1}$ است. چنانچه توابع پایه حقیقی مورد نظر باشد با استفاده از رابط اویلر می‌توان توابع زیر را بدست آورد:

$$\begin{aligned}
&\exp(a_k x - b_k y)\cos(b_k x + a_k y),\\
&\exp(a_k x - b_k y)\sin(b_k x + a_k y),\\
&\exp(a_k x + b_k y)\cos(b_k x - a_k y),\\
&\exp(a_k x + b_k y)\sin(b_k x - a_k y),\\
&\exp(-b_k x + a_k y)\cos(a_k x + b_k y),\\
&\exp(-b_k x + a_k y)\sin(a_k x + b_k y),\\
&\exp(b_k x + a_k y)\cos(a_k x - b_k y),\\
&\exp(b_k x + a_k y)\sin(a_k x - b_k y)
\end{aligned} \qquad (۲۵)$$

که در آن a_k و b_k دو عدد حقیقی هستند.

در تمامی مسائل دامنه حل به صورت $\Omega = [-1,1]\times[-1,1]$ فرض شده است. شرایط مرزی به صورت ضروری در نظر گرفته شده و از روی حل دقیق داده شده محاسبه شده است. مقادیر a_k و b_k در محدوده $[-1,1]$ با فواصل مساوی 0.05 در نظر گرفته شده است.

1 basic linear algebra sub-programs
2 free/open source
3 closed source
4 AMD core math library
5 Intel math kernel library
6 automatically tuned linear algebra software
7 open basic linear algebra sub-programs

جدول ۱ – مشخصات سیستم‌های رایانه‌ای و بسترهای نرم‌افزاری مورد استفاده

System	CPU		RAM	OS	C++ compiler	Mathematica
	Type	Cores				
1	Intel Core2 Duo P8600	2	4 GB DDR2	Ubuntu 12.10 (64 bit Linux)	GCC 4.6.3	9.0.1 (64 bit)
2	AMD Phenom Quad core 9950	4	4 GB DDR2	Ubuntu 12.10 (64 bit Linux)	GCC 4.6.3	9.0.1 (64 bit)
3	Intel Core i7 2700	4 + 4	8 GB DDR3	Ubuntu 12.10 (64 bit Linux)	GCC 4.6.3	9.0.1 (64 bit)

Table 1 – Specifications of hardware systems and software platforms

جدول ۲ – مقایسه کارایی روش‌های مختلف در ساخت ماتریس ضرایب روی سیستم شماره ۱

	Optimization performed	Coefficient matrix dimensions							
		80×3528		160×3528		80×13448		160×13448	
		Run time (s)	Speed up	Run time (s)	Speed up	Run time (s)	Speed up	Run time (s)	Speed up
Mathematica	None	1.50	1.00	2.97	1.00	5.56	1.00	11.03	1.00
	ParallelTable	1.25	1.20	2.94	1.01	5.16	1.08	8.77	1.26
	Compile	0.29	5.11	0.56	5.34	1.08	5.16	2.05	5.39
	Compile and ParallelTable	0.30	5.09	0.56	5.32	1.03	5.42	1.99	5.54
C++	None	0.12	12.32	0.19	14.84	0.34	15.40	0.59	17.90
	OpenMP	0.07	19.44	0.11	26.21	0.21	25.27	0.33	31.62

Table 2 – Comparison of performance in building coefficient matrix on System 1

جدول ۳ – مقایسه کارایی روش‌های مختلف در ساخت ماتریس ضرایب روی سیستم شماره ۲

	Optimization performed	Coefficient matrix dimensions							
		80×3528		160×3528		80×13448		160×13448	
		Run time (s)	Speed up	Run time (s)	Speed up	Run time (s)	Speed up	Run time (s)	Speed up
Mathematica	None	1.45	1.00	2.83	1.00	5.30	1.00	10.57	1.00
	ParallelTable	0.55	2.62	0.97	2.92	1.83	2.89	3.49	3.03
	Compile	0.27	5.48	0.50	5.65	0.95	5.60	1.82	5.81
	Compile and ParallelTable	0.28	5.28	0.51	5.54	0.95	5.57	1.82	5.80
C++	None	0.14	10.39	0.21	13.22	0.40	13.36	0.73	14.51
	OpenMP	0.05	28.81	0.07	39.64	0.12	44.31	0.21	51.12

Table 3 – Comparison of performance in building coefficient matrix on System 2

جدول ۴ – مقایسه کارایی روش‌های مختلف در ساخت ماتریس ضرایب روی سیستم شماره ۳

	Optimization performed	Coefficient matrix dimensions							
		80×3528		160×3528		80×13448		160×13448	
		Run time (s)	Speed up	Run time (s)	Speed up	Run time (s)	Speed up	Run time (s)	Speed up
Mathematica	None	0.71	1.00	1.38	1.00	2.64	1.00	5.26	1.00
	ParallelTable	0.32	2.19	0.46	2.98	0.82	3.22	1.58	3.33
	Compile	0.14	5.10	0.26	5.38	0.48	5.48	0.91	5.78
	Compile and ParallelTable	0.13	5.30	0.25	5.45	0.46	5.71	0.89	5.93
C++	None	0.07	10.09	0.10	14.16	0.18	14.98	0.27	19.22
	OpenMP	0.03	27.77	0.03	52.83	0.05	57.61	0.07	72.26

Table 4 – Comparison of performance in building coefficient matrix on System 3

جدول (۱) مشخصات سیستم‌های رایانه‌ای و بسترهای نرم‌افزاری مورد استفاده در این پژوهش را نشان می‌دهد. تمامی تست‌ها روی سیستم عامل لینوکس ٦٤ بیتی انجام شده است. همچنین از کامپایلر C++ همراه سیستم عامل (GCC) و نیز متمتیکای ٦٤ بیتی استفاده شده است.

جدول‌های ۲ تا ٤ کارایی روش‌های مختلف در ساخت ماتریس ضرایب را روی سیستم‌های مختلف نشان می‌دهد. برای هر یک از ابعاد نشان داده شده، زمان ساخت ماتریس و نیز سرعت نسبی محاسبه نسبت به پیاده‌سازی متمتیکا بدون بهینه‌سازی با همان ابعاد ماتریس ضرایب ذکر شده است. لازم به یادآوری است ساخت ماتریس ضرائب مستقل از حل دقیق می‌باشد. نگاهی به نتایج نشان می‌دهد در پیاده‌سازی متمتیکا، استفاده از دستور Compile می‌تواند زمان اجرا را بین ۵ تا ٦ برابر کاهش دهد. همچنین استفاده از دستور ParallelTable، با چشم‌پوشی از استفاده یا عدم استفاده از دستور Compile، گاهی تاثیر منفی ناچیز و گاهی تاثیر مثبت دارد. در مجموع و با در نظر گرفتن دقت‌های لازم در استفاده از دستور ParallelTable کاربرد آن توصیه نمی‌شود.

واضح است که در پیاده‌سازی C++ سرعت ساخت ماتریس ضرائب بسیار بالاتر خواهد بود. در پیاده‌سازی بدون بهینه‌سازی C++، افزایش سرعت نسبت به متمتیکا بدون بهینه‌سازی بین ۱۰ تا ۲۰ برابر و با استفاده از OpenMP بین ۲۰ تا ۷۰ برابر خواهد بود که نشان از برتری C++ در این قسمت از حل دارد.

جدول‌های (۵ تا ۷) کارایی روش‌های مختلف در حل دستگاه معادلات خطی را روی سیستم‌های مختلف نشان می‌دهد.

برای هر یک از ابعاد نشان داده شده، زمان حل دستگاه معادلات خطی و نیز سرعت نسبی محاسبه نسبت به پیاده‌سازی متمتیکا به وسیله PseudoInverse با همان ابعاد ماتریس ضرایب گفته شده است. در اینجا از تابع $u = \sin 2x \cosh 2y$ به عنوان حل دقیق استفاده شده است. در روند بدست آوردن نتایج، استفاده از کتابخانه توابع Intel MKL به نتایج عددی نادرستی منجر شد و بنابراین نتایج مربوط به آن گزارش نشده است. نتایج نشان می‌دهد عموماً استفاده از دستور PseudoInverse سریع‌تر از استفاده از دستورات SingularValueDecomposition و LinearSolve است، اما خطای آن بسیار بیشتر است.

جدول ۵ – مقایسه کارایی روش‌های مختلف در حل دستگاه معادلات خطی روی سیستم شماره ۱

	Implementation used	Coefficient matrix dimensions							
		80×3528		160×3528		80×13448		160×13448	
		Run time (s)	Speed up	Run time (s)	Speed up	Run time (s)	Speed up	Run time (s)	Speed up
Mathematica	PseudoInverse	0.07	1.00	0.16	1.00	0.34	1.00	1.21	1.00
	SingularValueDecomposition	0.10	0.68	0.21	0.76	0.37	0.92	1.28	0.94
	LinearSolve	0.07	0.93	0.13	1.30	0.33	1.03	0.87	1.38
C++	DGESDD (BLAS/LAPACK)	0.14	0.50	0.71	0.23	1.20	0.29	4.01	0.30
	DGESDD (ACML)	0.09	0.75	0.25	0.67	0.72	0.48	1.84	0.66
	DGESDD (OpenBLAS/LAPACK)	0.07	0.97	0.31	0.53	0.75	0.46	2.07	0.58
	DGELSD (BLAS/LAPACK)	0.05	1.36	0.30	0.55	0.58	0.59	1.97	0.61
	DGELSD (ACML)	0.04	1.67	0.11	1.52	0.38	0.89	1.03	1.17
	DGELSD (OpenBLAS/LAPACK)	0.04	1.84	0.14	1.14	0.35	0.99	1.06	1.14

Table 5 – Comparison of performance in solving system of linear equations on System 1

جدول ۶ – مقایسه کارایی روش‌های مختلف در حل دستگاه معادلات خطی روی سیستم شماره ۲

	Implementation used	Coefficient matrix dimensions							
		80×3528		160×3528		80×13448		160×13448	
		Run time (s)	Speed up	Run time (s)	Speed up	Run time (s)	Speed up	Run time (s)	Speed up
Mathematica	PseudoInverse	0.21	1.00	0.09	1.00	0.19	1.00	0.66	1.00
	SingularValueDecomposition	0.07	2.82	0.14	0.62	0.29	0.65	0.90	0.74
	LinearSolve	0.06	3.71	0.15	0.58	0.33	0.58	0.91	0.72
C++	DGESDD (BLAS/LAPACK)	0.16	1.32	0.63	0.14	0.90	0.21	3.15	0.21
	DGESDD (ACML)	0.26	0.79	0.34	0.26	0.66	0.29	1.62	0.41
	DGESDD (OpenBLAS/LAPACK)	0.12	1.71	0.37	0.24	0.55	0.34	1.57	0.42
	DGELSD (BLAS/LAPACK)	0.08	2.52	0.30	0.30	0.47	0.41	1.48	0.45
	DGELSD (ACML)	0.07	3.00	0.16	0.54	0.36	0.53	0.89	0.74
	DGELSD (OpenBLAS/LAPACK)	0.06	3.73	0.18	0.50	0.28	0.69	0.78	0.85

Table 6 – Comparison of performance in solving system of linear equations on System 2

جدول ۷ – مقایسه کارایی روش‌های مختلف در حل دستگاه معادلات خطی روی سیستم شماره ۳

	Implementation used	Coefficient matrix dimensions							
		80×3528		160×3528		80×13448		160×13448	
		Run time (s)	Speed up	Run time (s)	Speed up	Run time (s)	Speed up	Run time (s)	Speed up
Mathematica	PseudoInverse	0.02	1.00	0.04	1.00	0.05	1.00	0.22	1.00
	SingularValueDecomposition	0.03	0.73	0.05	0.80	0.08	0.67	0.30	0.73
	LinearSolve	0.03	0.71	0.05	0.74	0.11	0.51	0.26	0.83
C++	DGESDD (BLAS/LAPACK)	0.06	0.32	0.20	0.19	0.28	0.19	1.14	0.19
	DGESDD (ACML)	0.05	0.40	0.13	0.29	0.24	0.23	0.69	0.32
	DGESDD (OpenBLAS/LAPACK)	0.04	0.54	0.07	0.51	0.12	0.46	0.51	0.42
	DGELSD (BLAS/LAPACK)	0.02	0.86	0.08	0.49	0.12	0.45	0.51	0.42
	DGELSD (ACML)	0.02	0.91	0.06	0.66	0.12	0.45	0.35	0.62
	DGELSD (OpenBLAS/LAPACK)	0.01	2.13	0.02	1.55	0.04	1.35	0.22	1.00

Table 7 – Comparison of performance in solving system of linear equations on System 3

جدول ۸ – مقایسه خطای حاصل از روش‌های مختلف روی سیستم شماره ۱ بر حسب شاخص خطای e_{L_2}

	Implementation used	Exact solution	Coefficient matrix dimensions			
			80×3528	160×3528	80×13448	160×13448
Mathematica	PseudoInverse	#1	8.7418E-06	9.4599E-06	5.2358E-06	3.7430E-06
		#2	8.7418E-06	9.4599E-06	5.2358E-06	3.7430E-06
	SingularValueDecomposition	#1	1.0026E-08	1.0132E-08	9.1609E-09	9.2263E-09
		#2	1.0026E-08	1.0132E-08	9.1609E-09	9.2263E-09
	LinearSolve	#1	1.0028E-08	1.0125E-08	9.4706E-09	9.2283E-09
		#2	1.0028E-08	1.0125E-08	9.4706E-09	9.2283E-09
C++	DGESDD (BLAS/LAPACK)	#1	2.6508E-11	2.2198E-11	2.6123E-11	2.2272E-11
		#2	1.0041E-08	1.0133E-08	9.1849E-09	9.2362E-09
	DGESDD (ACML)	#1	2.5897E-11	2.1661E-11	2.8791E-11	2.2790E-11
		#2	1.0042E-08	1.0134E-08	9.3201E-09	9.4802E-09
	DGESDD (OpenBLAS/LAPACK)	#1	2.5860E-11	2.1907E-11	2.7674E-11	2.2449E-11
		#2	1.0037E-08	1.0141E-08	9.1429E-09	9.3075E-09
	DGELSD (BLAS/LAPACK)	#1	2.5914E-11	2.1634E-11	2.5422E-11	2.1493E-11
		#2	1.0032E-08	1.0132E-08	9.2808E-09	9.2479E-09
	DGELSD (ACML)	#1	1.9893E-06	2.1943E-11	2.7645E-11	2.1554E-11
		#2	1.0492E-08	1.0136E-08	9.2543E-09	9.2669E-09
	DGELSD (OpenBLAS/LAPACK)	#1	2.5967E-11	2.1703E-11	2.5816E-11	2.1467E-11
		#2	1.0042E-08	1.0135E-08	9.1588E-09	9.2264E-09

Table 8 – Comparison of error of different methods in e_{L_2} norm on System 1

جدول ۹ – مقایسه خطای حاصل از روش‌های مختلف روی سیستم شماره ۲ بر حسب شاخص خطای e_{L_2}

	Implementation used	Exact solution	Coefficient matrix dimensions			
			80×3528	160×3528	80×13448	160×13448
Mathematica	PseudoInverse	#1	9.3644E-06	3.1388E-06	1.4709E-06	2.6977E-06
		#2	1.6122E-05	4.6128E-06	1.0943E-05	4.5888E-06
	SingularValueDecomposition	#1	2.6029E-11	2.1639E-11	2.5943E-11	2.1350E-11
		#2	1.0036E-08	1.0129E-08	9.1396E-09	9.2346E-09
	LinearSolve	#1	1.5682E-09	1.2997E-09	1.5449E-09	1.2803E-09
		#2	1.0027E-08	1.0130E-08	9.1440E-09	9.2355E-09
C++	DGESDD (BLAS/LAPACK)	#1	2.6508E-11	2.2198E-11	2.6123E-11	2.2272E-11
		#2	1.0041E-08	1.0133E-08	9.1849E-09	9.2362E-09
	DGESDD (ACML)	#1	2.6005E-11	2.1584E-11	2.9833E-11	2.3098E-11
		#2	1.0037E-08	1.0145E-08	9.3374E-09	9.4874E-09
	DGESDD (OpenBLAS/LAPACK)	#1	2.5864E-11	2.1627E-11	2.7638E-11	2.2855E-11
		#2	1.0041E-08	1.0140E-08	9.1429E-09	9.2595E-09
	DGELSD (BLAS/LAPACK)	#1	2.5914E-11	2.1634E-11	2.5422E-11	2.1493E-11
		#2	1.0032E-08	1.0132E-08	9.2808E-09	9.2479E-09
	DGELSD (ACML)	#1	2.7068E-11	2.1746E-11	2.8820E-11	2.9326E-05
		#2	1.0030E-08	1.0141E-08	9.2691E-09	1.5610E-06
	DGELSD (OpenBLAS/LAPACK)	#1	2.5995E-11	2.1582E-11	2.5714E-11	2.1627E-11
		#2	1.0037E-08	1.0131E-08	9.1848E-09	1.0420E-08

Table 9 – Comparison of error of different methods in e_{L_2} norm on System 2

جدول ۱۰ – مقایسه خطای حاصل از روش‌های مختلف روی سیستم شماره ۳ بر حسب شاخص خطای e_{L_2}

	Implementation used	Exact solution	Coefficient matrix dimensions			
			80×3528	160×3528	80×13448	160×13448
Mathematica	PseudoInverse	#1	3.8229E-06	3.1323E-06	2.4107E-06	1.3534E-06
		#2	6.6210E-06	9.8172E-06	1.7858E-06	4.9967E-06
	SingularValueDecomposition	#1	2.5791E-11	2.1571E-11	2.6685E-11	2.1674E-11
		#2	1.0025E-08	1.0130E-08	9.1620E-09	9.2261E-09
	LinearSolve	#1	1.5682E-09	1.2997E-09	1.5448E-09	1.2803E-09
		#2	1.0028E-08	1.0143E-08	1.0181E-08	9.5210E-09
C++	DGESDD (BLAS/LAPACK)	#1	2.6537E-11	2.2178E-11	2.6193E-11	2.2258E-11
		#2	1.0033E-08	1.0131E-08	9.1655E-09	9.2354E-09
	DGESDD (ACML)	#1	2.5947E-11	2.1565E-11	2.9739E-11	2.3198E-11
		#2	1.0044E-08	1.0131E-08	9.2597E-09	9.4483E-09
	DGESDD (OpenBLAS/LAPACK)	#1	2.5967E-11	2.1871E-11	2.7667E-11	2.3056E-11
		#2	1.0029E-08	1.0135E-08	9.1750E-09	9.3717E-09
	DGELSD (BLAS/LAPACK)	#1	2.6033E-11	2.1752E-11	2.5609E-11	2.1621E-11
		#2	1.0029E-08	1.0135E-08	9.2572E-09	9.2365E-09
	DGELSD (ACML)	#1	2.6291E-11	2.2513E-11	2.8719E-11	2.1403E-11
		#2	1.0029E-08	1.0360E-08	9.2212E-09	1.0186E-08
	DGELSD (OpenBLAS/LAPACK)	#1	2.5802E-11	2.3983E-11	2.5663E-11	2.2824E-11
		#2	1.0024E-08	1.0129E-08	9.1903E-09	9.2625E-09

Table 10 – Comparison of error of different methods in e_{L_2} norm on System 3

در نتیجه اگر سطح خطای یکسانی مورد انتظار باشد، استفاده از دو دستور گفته شده برتری قابل ملاحظه‌ای خواهند داشت. همچنین در کد C++ هنگامی که ابعاد ماتریس بزرگ‌تر می‌شود، کارایی روش‌های مختلف مورد استفاده نسبت به کد متمتیکا کاهش می‌یابد. این امر دور از انتظار نیست، زیرا دستورات کتابخانه‌ای متمتیکا بسیار بهینه شده‌اند و کارایی

بسیار بالایی از خود نشان می‌دهند. نگاهی به زمان‌های اجرا در دو بخش ساخت ماتریس ضرائب و حل دستگاه معادلات خطی نشان می‌دهد در کد متمتیکا قسمت اول بیشترین زمان را به خود اختصاص می‌دهد و بنابراین کارایی بسیار بالای کد C++ در این قسمت به راحتی ضعف آن را در قسمت دوم پوشش می‌دهد.

برای مقایسه دقت روش‌های مختلف، شاخص خطای e_{L_2} هر روش در جدول‌های (۸ تا ۱۰) نشان داده شده است. شاخص خطای e_{L_2} به صورت زیر تعریف می‌شود:

$$e_{L_2} = \sqrt{\sum_{j=1}^{\bar{n}} \left(\hat{u}\big|_{\bar{\mathbf{x}}_j} - u\big|_{\bar{\mathbf{x}}_j} \right)^2 \Big/ \sum_{j=1}^{\bar{n}} \left(\hat{u}\big|_{\bar{\mathbf{x}}_j} \right)^2} \qquad (۲۶)$$

که در آن $\bar{n}$ تعداد نقاط یک شبکه روی کل دامنه به فاصله برابر با نقاط روی مرز و $\bar{\mathbf{x}}_j$ یکی از نقاط آن می‌باشد. همچنین از تابع $u = \sin 2x \cosh 2y$ به عنوان حل دقیق اول و از تابع $u = x^{10} - 45x^8y^2 + 210x^6y^4 - 210x^4y^6 + 45x^2y^8 - y^{10}$ به عنوان حل دقیق دوم استفاده شده است. نتایج نشان می‌دهد در پیاده‌سازی متمتیکا استفاده از دستور PseudoInverse کمترین دقت را دارا است. همچنین دستورات LinearSolve و SingularValueDecomposition به ترتیب دقت‌های بیشتری را ارائه می‌دهند. در برنامه C++ تمامی روش‌های به کار برده شده حداکثر دقت بدست آمده در برنامه متمتیکا را ارائه می‌کنند.

با توجه به مجموع نتایج ارائه شده می‌توان گفت در صورتی که از برنامه متمتیکا برای پیاده‌سازی روش توابع پایه نمایی استفاده شود، بهتر است از دستور Compile برای ساخت ماتریس ضرائب و از دستور SingularValueDecomposition برای حل دستگاه معادلات نهایی استفاده شود. همچنین در برنامه‌هایی که به زبان C++ (یا سایر زبان‌های مشابه مانند فرترن) نوشته می‌شوند، بهتر است از OpenMP برای تسریع در ساخت ماتریس ضرائب استفاده شود. برای حل دستگاه معادلات نهایی نیز تابع DGELSD از کتابخانه توابع LAPACK همراه با کتابخانه OpenBLAS توصیه می‌شود. در صورت استفاده از این توابع که از نظر دقت و تا حدی از نظر سرعت بهینه هستند، برنامه C++ بین ۲/۵ تا ٦ برابر سریع‌تر از برنامه متمتیکا است. این موضوع نشان می‌دهد اگر چه برنامه متمتیکا و سایر بسته‌های نرم‌افزاری مشابه برای توسعه روش‌ها بسیار مفید هستند، نمی‌توان در مسائل بزرگ از آنها استفاده نمود. همچنین باید دانست که برنامه‌های نوشته شده به وسیله‌ی کاربر بسیار کندتر از توابع کتابخانه‌ای خواهند بود. این امر از این لحاظ مهم است که برای مقایسه یک روش تازه توسعه داده شده با روش‌های موجود از نظر کارایی، ممکن است استفاده از این بسته‌ها به نتیجه‌گیری‌های غلطی منجر شود.

٦- جمع‌بندی و نتیجه‌گیری

در این پژوهش پیاده‌سازی روش توابع پایه نمایی روی دو بستر نرم‌افزاری متمتیکا به عنوان یک نمونه از بسته‌های نرم‌افزاری ریاضی و C++ به عنوان یک زبان برنامه‌نویسی عادی مورد بررسی قرار گرفت. نتایج بدست آمده روی معادله لاپلاس روی سیستم‌های مختلف نشان داد که برای ساخت بهینه ماتریس ضرائب می‌توان در متمتیکا از دستور Compile و در C++ از OpenMP استفاده نمود. همچنین در حل دستگاه معادلات دیفرانسیل استفاده از دستور SingularValueDecomposition از نظر خطا برتری زیادی نسبت به دستور PseudoInverse دارد. همچنین در برنامه C++ کاربرد دستور DGELSD از کتابخانه توابع LAPACK همراه با کتابخانه توابع OpenBLAS توصیه می‌شود. در حالت بهینه، کارایی برنامه C++ در نمونه‌های بررسی شده بین ۲/۵ تا ٦ برابر برنامه متمتیکا است.

٦- مراجع

References

[1] Zienkiewicz, O. C., "Achievements and some unsolved problems of the finite element method", International Journal for Numerical Methods in Engineering, Vol. 47, pp. 9-28, 2000.

[2] Boroomand, B., Soghrati, S. and Movahedian, B., "Exponential basis functions in solution of static and time harmonic elastic problems in a meshless style", International Journal for Numerical Methods in Engineering, Vol. 81, pp. 971-1018, 2010.

[3] Shahbazi, M., Boroomand, B. and Soghrati, S., "A mesh-free method using exponential basis functions for laminates modeled by CLPT, FSDT and TSDT. Part I: Formulation", Composite Structures, Vol. 93, pp. 3112-

3119, 2011.
[4] Shahbazi, M., Boroomand, B. and Soghrati, S., "A mesh-free method using exponential basis functions for laminates modeled by CLPT, FSDT and TSDT. Part II: Implementation and results", Composite Structures, Vol. 94, pp. 84-91, 2011.
[5] Shamsaei, B. and Boroomand, B., "Exponential basis functions in solution of laminated structures", Composite Structures, Vol. 93, pp. 2010-2019, 2011.
[6] Shahbazi, M., Boroomand, B. and Soghrati, S., "On using exponential basis functions for laminates modeled by CLPT, FSDT and TSDT: further tests and results", Composite Structures, Vol. 94, pp. 2263-2268, 2012.
[7] Zandi, S. M., Boroomand, B. and Soghrati, S., "Exponential basis functions in solution of problems with fully incompressible materials: A mesh-free method", Journal of Computational Physics, Vol. 231, pp. 7255-7273, 2012.
[8] Zandi, S. M., Boroomand, B. and Soghrati, S., "Exponential basis functions in solution of incompressible fluid problems with moving free surfaces", Journal of Computational Physics, Vol. 231, pp. 505-527, 2012.
[9] Azhari, F., Boroomand, B. and Shahbazi, M., "Explicit relations for the solution of laminated plates modeled by a higher shear deformation theory: Derivation of exponential basis functions", International Journal of Mechanical Sciences, Vol. 77, pp. 301-313, 2013.
[10] Azhari, F., Boroomand, B. and Shahbazi, M., "Exponential basis functions in the solution of laminated plates using a higher order ZigZag theory", Composite Structures, Vol. 105, pp. 398-407, 2013.
[11] Boroomand, B., Azhari, F. and Shahbazi, M., "On definition of clamped conditions in TSDT and FSDT; the use of exponential basis functions in solution of laminated composites", Composite Structures, Vol. 97, pp. 129-135, 2013.
[12] Hashemi, S. H., Boroomand, B. and Movahedian, B., "Exponential basis functions in space and time: A meshless method for 2D time dependent problems", Journal of Computational Physics, Vol. 241, pp. 526-545, 2013.
[13] Movahedian, B., Boroomand, B. and Soghrati, S., "A Trefftz method in space and time using exponential basis functions: Application to direct and inverse heat conduction problems", Engineering Analysis with Boundary Elements, Vol. 37, pp. 868-883, 2013.
[14] Movahedian, B. and Boroomand, B., "The solution of direct and inverse transient heat conduction problems with layered materials using exponential basis functions", International Journal of Thermal Sciences, Vol. 77, pp. 186-198, 2014.
[15] Abdollahi, R. and Boroomand, B., "Nonlocal elasticity defined by Eringens integral model: Introduction of a boundary layer method", International journal of solids and structures, Vol. 51, pp. 1758-1780, 2014.
[16] Ben-Israel, A., and Greville, T. N. E., Generalized inverses, Springer, 2003.
[17] Linear Algebra Package (LAPACK), http://netlib.org/lapack.
[18] Basic Linear Algebra Subprograms (BLAS), http://netlib.org/blas
[19] AMD Core Math Library (ACML), http://developer.amd.com/tools-and-sdks/cpu-development
[20] Intel Math Kernel Library (MKL), https://software.intel.com/en-us/intel-mkl
[21] Automatically Tuned Linear Algebra Software (ATLAS), http://netlib.org/atlas
[22] OpenBlas, http://openblas.net

Optimal Implementation of Exponential Basis Functions Method on Different Software Platforms and Performance Comparison

F. Mossaiby[1*]

1- Assist. Prof., Department of Civil and Transportation Engineering, University of Isfahan

***mossaiby@eng.ui.ac.ir**

Abstract:

Despite the success and versatility of mesh based methods --finite element method in particular- there has been a growing demand in last decades towards the development and adoption of methods which can eliminate using the mesh, i.e. the so called meshless or mesh-free methods. Difficulties in the generation of high quality meshes, in terms of computational cost, technical problems such as serial nature of the mesh generation process and the urge of parallel processing for today's huge problems have been the main motivation for the implementation of new researches. Apart from these, the human required expertise can never be completely omitted from the analysis process. However, the problem is much more pronounced in 3D problems. To this end, many meshless methods have been developed in recent years among which SPH, EFG, MLPG, RKPM, FPM and RBF-based methods could be named. The exponential basis functions method (EBF) is one of these methods which has been successfully employed in various engineering problems, ranging from heat transfer and various plate theories to classical and non-local elasticity and fluid dynamics. The method uses a linear combination of exponential basis functions to approximate the field variables. It is shown that these functions have very good approximation capabilities and their application guarantees a high convergence rate. These exponential bases are chosen such that they satisfy the homogenous form of the differential equation. This leads to an algebraic characteristic equation in terms of exponents of basic functions. From this point of view, this method may be categorized as an extension to the well-known Trefftz family of methods. These methods rely on a set of the so called T-complete bases for their approximation of the field variables. These bases should satisfy the homogenous form of the governing equation. They have been used with various degrees of success in a wide range of problems. The main drawback of these methods –however- lies in the determination of the basis, which should be found for every problem. This problem has been reduced to the solution of the algebraic characteristic equation in the exponential basis functions method. The method is readily applicable to linear, constant coefficient operators, and has been recently extended to more general cases of linear and also non-linear problems with variable coefficients. The relative performance of usual programming languages such as C++ in comparison with mathematical software packages -like Mathematica and/or Matlab- is one of the major questions when using such packages to develop new numerical methods. This can affect the interpretation of the performance of newly developed methods compared to established ones. In this paper, the implementation of the exponential basis functions method on various software platforms has been discussed. C++ and Mathematica programming have been examined as a representative of different software platforms. The exponential basis function method is implemented in each platform, using various options available. Results show that with a proper implementation, the numerical error of the method can be decreased considerably. Regarding the results of this research, optimal implementations of C++ and Mathematica platforms, error ratio is between 2.5 and 6, respectively.

Keywords: Exponential basis function method (EBF), Pseudo-Inverse, Singular value decomposition (SVD), Partial differential equations (PDE), Optimal implementation