Notizen zu Numerische Methoden, Serie 7

David Nadlinger

Numerische Methoden für D-PHYS, FS 2013, ETH Zürich

LINEARE ODES MIT KRYLOV-VERFAHREN

Das Ziel dieser Aufgabe ist es, ein System linearer Differentialgleichungen zu lösen, also bei Kenntnis der Matrix $\mathbf{B} \in \mathbb{C}^{d\times d}$ und der Zusammenhänge

$$\dot{\mathbf{y}}(t) = \mathbf{B}\,\mathbf{y}(t) \tag{1}$$
$$\mathbf{y}(0) = \mathbf{y}_0 \tag{2}$$

den Vektor $\mathbf{y}(t)$ zu allen Zeiten t zu bestimmen. Die allgemeine Lösung von Differentialgleichungen dieses Typs ist aus der Analysis bekannt:

$$\mathbf{y}(t) = \mathrm{e}^{\mathbf{B}t}\mathbf{y}_0. \tag{3}$$

Wenn euch diese Formulierung noch nicht vertraut vorkommt, wäre dies wahrscheinlich ein guter Zeitpunkt, um sich noch einmal damit auseinanderzusetzen, da ihr in Linearer Algebra, Analysis, aber auch Physik noch des öfteren damit zu tun haben werdet. Stellt sicher, dass ihr die Theorie verstanden habt, und passt die Gleichungen an die zusätzlichen Faktoren im Beispiel an – schon habt ihr Punkt b) erledigt.

Unitäre Evolution. In dieser Aufgabe ist noch zusätzlich bekannt, dass $\mathbf{A}$ Hermite-symmetrisch ist. Ihr sollt nun in einem ersten Schritt zeigen, dass sich als Konsequenz daraus die $\|\cdot\|_2$-Norm des Vektors mit der Zeit nicht ändert. Dazu habt ihr mehrere Möglichkeiten: Am direktesten berechnet ihr $\frac{\mathrm{d}}{\mathrm{d}t}\|\mathbf{y}(t)\|_2^2$ explizit und zeigt, dass der Ausdruck verschwindet (das Quadrat zu betrachten vermeidet die störende Wurzel, ein öfter recht praktischer Trick).

Ihr könnt aber auch versuchen, die Differenz der Normen $\|\mathbf{y}(t)\|_2$ und $\|\mathbf{y}(0)\|_2$ auszuschreiben (unter Zuhilfenahme von (1)), oder allgemein zu zeigen, dass das Exponential einer hermiteschen Matrix unitär ist.

Überlegungen zur numerischen Lösung. Prinzipiell gibt Gleichung (1) schon die komplette Lösung an. Allerdings benötigen wir dafür das Exponential der Matrix $-\mathrm{i}\mathbf{A}t$. Für kleinere Matrizen ist dieses auch recht gut direkt berechenbar, etwa mit `scipy.linalg.expm`, das intern mit einer Padè-Approximation arbeitet. Hat man es mit grösseren Matrizen zu tun (grössenordnungsmässig *in etwa* $d \geq 50$), insbesondere mit dünnbesetzten Matrizen, ist diese Vorgehensweise aber nicht mehr optimal, da der Rechenaufwand schnell ansteigt.

Was also tun? Nehmen wir an, die Matrix $\mathbf{A}$ sei diagonalisierbar, also $\exists \mathbf{S} \in \mathrm{GL}_{\mathbb{C}}(d), \mathbf{D} = \mathrm{diag}((\lambda_k)_k) : \mathbf{S}\,\mathbf{A}\,\mathbf{S}^{-1} = \mathbf{D}$, wobei $(\lambda_k)_{k=1,\dots,d} \subset \mathbb{C}$ die Eigenwerte von $\mathbf{A}$ sind und die zugehörigen Eigenvektoren in den Spalten von $\mathbf{S}$ stehen (warum ist dies für $\mathbf{A}$ im Beispiel tatsächlich der Fall?). In diesem Fall gilt $\mathrm{e}^{\mathbf{A}} = \mathbf{S}\,\mathrm{e}^{\mathbf{D}}\,\mathbf{S}^{-1} = \mathbf{S}\,\mathrm{diag}((\mathrm{e}^{\lambda_k})_k)\,\mathbf{S}^{-1}$. (Auch das ist wieder eine Gleichung, deren Beweis es sich wirklich zu verstanden haben lohnt.) Kennen wir also die Eigenwerte und zugehörigen Eigenvektoren, können wir auch das Matrizenexponential berechnen, wie ihr es in Analysis oder Linearer Algebra vielleicht schon das eine oder andere Mal händisch getan habt.

Wie hilft uns dieser Zugang weiter? Erinnern wir uns: Einer der Vorteile der Potenzmethode ist ja, dass sie bloss auf der Matrix-Vektor-Multiplikation $\mathbf{A}\,\mathbf{y}$ aufbaut, und nicht direkt etwa auf die Einträge der Matrix zugreift, daher also auch für dünnbesetzte Matrizen gut geeignet ist. Allerdings liefert sie bloss den betragsgrössten Eigenwert (oder über die inverse Potenzmethode vielleicht auch noch den kleinsten). Wie kommen wir an die anderen Eigenwerten und Eigenvektoren heran?

Nun, eine Idee wäre, zunächst einmal mittels der Potenzmethode das erste Eigenwert-Eigenvektor-Paar zu bestimmen. Im Anschluss könnten wir die Iteration mehrmals wiederholen, aber jetzt in jedem Schritt dafür sorgen, dass unsere Vektoren immer senkrecht auf allen bereits bestimmten Eigenvektoren stehen (analog zum Gram-Schmidt-Verfahren jeweils die Projektion abziehen, …). Wir könnten vermuten, dass dieses Verfahren zum jeweils nächstgrössten Eigenwert konvergiert – und hätten Recht, zumindest wenn $\mathbf{A}$ unitär diagonalisierbar ist, da ja die Eigenvektoren dann ein Orthogonalsystem bilden.

Diese Vorgehensweise ist aber in zweierlei Hinsicht nicht optimal: Einerseits ist das Gram-Schmidt-Verfahren numerisch ziemlich instabil, wie ihr schon in einer der vorigen Serien nachprüfen durftet. Andererseits verwenden wir immer nur den letzten Wert jeder Iteration, und werfen alle Zwischenergebnisse weg. Wie können wir das Verfahren anpassen, um diese Nachteile zu umgehen?

Krylov-Unterraumverfahren. Gehen wir noch einmal einen Schritt zurück: Die Idee war, die teure direkte Berechnung des Matrizenexponentials zu vermeiden. Aber halt: Dies gilt ja nur für *grosse* Matrizen, für kleinere Matrizen haben wir durchaus adäquate Methoden zu Verfügung. Können wir die Berechnung unseres Ergebnisses vielleicht durch das Exponential einer kleineren Matrix annähern?

Die Antwort auf diese Frage lautet bemerkenswerterweise «ja», und es handelt sich dabei um die zentrale Idee der *Krylov-Unterraumverfahren*, von denen wir in diesem Kurs die *Arnoldi-Iteration* und als spezialisierte Variante davon das *Lanczos-Verfahren* behandeln. Die Theorie dazu ist im Skript und den alten Folien zur Vorlesung eigentlich recht ausführlich dargestellt, inklusive Python-Code zur Implementierung der Verfahren. Daher möchte ich hier nur die wichtigsten Punkte erwähnen.

Die Arnoldi-Iteration hat neben einem Startvektor $\mathbf{y}_0$ als Parameter die Dimension $k \leq d$ (im den Vorlesungunterlagen auch oft m) des zu verwendenden Krylovraums, und erzeugt für eine gegebene Matrix $\mathbf{A}$ zwei Ergebnisse, eine Basis $\mathbf{V}_k \in \mathbb{C}^{d\times k}$ des Krylov-Unterraums $K_k(\mathbf{A}, \mathbf{y}_0)$ und eine obere Hessenberg-Matrix $\mathbf{H}_k \in \mathbb{C}^{k\times k}$ (sofern das Verfahren nicht vorzeitig terminiert – Division durch Null). Da wir ja die Dimensionen der Matrix, die wir zu exponentieren haben, verringern wollen, ist k sinnvollerweise deutlich kleiner als d. In diesem Fall ist $\mathbf{V}_k$ eine grosse schmale und $\mathbf{H}_k$ eine kleine quadratische Matrix.

Für die erhaltenen Matrizen gilt nun

$$\mathbf{V}^\dagger \mathbf{V} = \mathbf{I}_k, \tag{4}$$

die Spalten sind also orthonormal, und

$$\mathbf{H}_k = \mathbf{V}_k^\dagger \mathbf{A} \mathbf{V}_k. \tag{5}$$

Ist $\mathbf{A}$ Hermite-symmetrisch, so ist $\mathbf{H}_k^\dagger = \mathbf{H}_k$ (warum?), und damit ist $\mathbf{H}_k$ tridiagonal mit identischen Einträgen auf der oberen und unteren Nebendiagonalen ($\mathbf{H}_k$ ist ja immer in Hessenberg-Form!). Diese Tatsache macht man sich zu Nutze, um das Arnoldi- zum Lanczos-Verfahren zu vereinfachen – $\mathbf{H}_k$ kann durch zwei Vektoren beschrieben werden.

Approximation des Matrixexponentials. Wir haben also $\mathbf{V}_k$ und $\mathbf{H}_k$ für eine gegebene Matrix $\mathbf{A}$ bestimmt. Wie hilft uns das aber bei der Lösung des ursprünglichen Problems, der Zeitevolution eines linearen Differentialgleichungssystems?

Nun, die approximierte Lösung $\mathbf{u}_k(t)$ soll ja in $K_k(\mathbf{A}, \mathbf{y}_0)$ liegen. Also können wir schreiben

$$\mathbf{u}_k(t) = \mathbf{V}_k \mathbf{x}(t), \tag{6}$$

wobei $\mathbf{x}(t) \in \mathbb{C}^k$. Die Forderung nach der Orthogonalität des Residuums ist äquivalent mit der Aussage, dass

$$\forall \mathbf{d} \in \mathbb{C}^k : \langle \mathbf{V}_k \mathbf{d}, \mathbf{V}_k \dot{\mathbf{x}}(t) + \mathrm{i}\, \mathbf{A}\, \mathbf{V}_k\, \mathbf{x}(t) \rangle = 0 \tag{7}$$

$$\Leftrightarrow \mathbf{V}_k^\dagger (\mathbf{V}_k \dot{\mathbf{x}}(t) + \mathrm{i}\, \mathbf{A}\, \mathbf{V}_k\, \mathbf{x}(t)) = 0. \tag{8}$$

Für $\mathbf{x}(t)$ ergibt sich also die Gleichung

$$\dot{\mathbf{x}}(t) = -\mathrm{i}\, \mathbf{V}_k^\dagger \mathbf{A}\, \mathbf{V}_k \mathbf{x}(t), \tag{9}$$

die ihr wie gehabt lösen könnt. Welche Matrix steht auf der rechten Seite? Wie ergibt sich daraus $\mathbf{u}_k(t) (\approx \mathbf{y}(t))$?

Auch hier ist das obige schon wieder der grösste Teil der Antwort zu Frage c). Versucht aber, die einzelnen Schritte im Detail nachzuvollziehen!

Tipps zur Implementierung. Das Herzstück eures Programms ist natürlich das Arnoldi- bzw. Lanczos-Verfahren. Für deren Implementierung könnt ihr euch an den Beispiel-Code im Skript halten (Code 4.3.22, Seite 89, und Code 4.3.27, Seite 91). Passt aber auf: Die Lanczos-Implementierung im Skript verwendet reelle Zahlen, während das Template mit komplexen Werten arbeitet. Passt die Arrays bei der Erzeugung entsprechend an (siehe die Implementierung des Arnoldi-Verfahrens).

Eine weitere potentielle Stolperfalle sind die Rückgabewerte der Funktionen: Lest euch den Code sorgfältig durch und versucht herauszufinden, ob es sich dabei $\mathbf{V}_k$ oder $\mathbf{V}_{k+1}$, $\mathbf{H}_k$ oder $\tilde{\mathbf{H}}_k$, etc. handelt. Im Zweifelsfall könnt ihr das gut über die Dimension der Arrays kontrollieren.

Sonst ist das Beispiel wenig aufregend. Für die Zeitmessung nehmt einfach `time.clock()` wie üblich, und für den Fehler betrachtet die Norm der Differenz zwischen dem jeweiligen Ergebnis und dem direkt mittels `expm` und Gleichung (1) erhaltenen Vektor.

Wenn ihr alles richtig gemacht habt, sollten die beiden Krylov-Verfahren um eine Größenordnung schneller sein als die SciPy-Funktionen, trotzdem aber gute Resultate liefern. Und wundert euch nicht: Eine der drei `expm`-Varianten versagt kläglich!

NEWTON-INTERPOLATION

Dieses Beispiel ist wie gesagt rein mit Papier und Bleistift zu bewältigen. Ihr sollt dabei zunächst ein Interpolationspolynom für die Quadratwurzel-Funktion mit drei gegebenen Stützstellen aufstellen. Dies funktioniert nach dem Verfahren der dividierten Differenzen, das in der Vorlesung behandelt wurde. Im Zweifelsfall könnt ihr noch einmal im Skript, Abschnitt 5.3.3 (Seite 106 ff.) nachschlagen.

Dann wendet ihr Theorem 5.3.11 (Skript Seite 108) auf $f(t) = \sqrt{t}$ und die gegebenen Stützstellen an, um eine Abschätzung für den Interpolationsfehler an der Stelle $t = 2$ zu erhalten. Im dritten Schritt berechnet ihr die tatsächliche Differenz, und vergleicht sie mit dem (zwangsläufig «pessimistischen») Schätzwert.

3-TERM-REKURSION

Auch dieses Beispiel solltet ihr ohne gröbere Probleme lösen können: Stellt die gegebene Rekursionsformel noch so um, dass sie in der richtigen Form für eine Rückwärtsrekursion ist, und implementiert beide Verfahren in einer kleinen Schleife.

Ihr solltet kaum mehr als 15–20 Zeilen Code benötigen. Überlegt euch vor allem noch etwas zur gestellten Theoriefrage – eventuell könnt ihr euch dazu auch noch an das ein oder andere Beispiel aus Informatik erinnern.

Viel Spass beim Lösen der Serie!