Ü 6 Sechste Übungseinheit

Inhalt der sechsten Übungseinheit:

- Polynomiale Interpolation
- Numerische Integration

Ü 6.1 Polynomiale Interpolation

Informieren Sie sich zu diesem Themenbereich: Folien der 6. Vorlesung und Kapitel 7 im Skriptum. Werfen Sie ruhig auch einen Blick auf den Wikipedia-Artikel zum Stichwort Polynominterpolation.

Häufig sind Werte einer Funktion tabellarisch gegeben, wie im folgenden Datensatz:

T	cp	
300	537.0	Spezifische Wärmekapazität von kohlenstoffarmem Stahl in J/kg K für Temperaturen zwischen 300 und 600C
400	593.3	
500	666.8	
600	760.8	

Tabelle 1: Musterdatensatz

Gesucht ist ein Polynom durch diese Datenpunkte. Mögliche Vorgangsweisen:

- Ansatz des Polynoms mit unbestimmten Koeffizienten: $p(x) = a + bx + cx^2 + dx^3$, Einsetzen der Datenpunkte, Aufstellen und Lösen des Gleichungssystems. Siehe Aufgabe 48.
- Lagrangesche Interpolationsformel. Kennen Sie aus der Mathematik-I-Vorlesung. Auch hier im Skriptum behandelt. Vorteil: Polynom lässt sich direkt hinschreiben; Änderung der y-Daten leicht möglich. Nachteil: nicht die günstigste Form zur rechnergestützten Auswertung.
- Das Skriptum erklärt zwei weitere Rechenschemen (Verfahren von Neville und Newton, dividierte Differenzen). Diese Methoden sind für das Rechnen von Hand optimiert und stammen aus der Zeit, bevor es moderne Computer gab. Rechnen von Hand ist heute nicht mehr so wichtig.
- Interpolation in der Newton-Form ist auch am Computer vorteilhaft, wenn es – etwa bei Echtzeit-Anwendungen – auf möglichst kurze Rechenzeit ankommt. Diese Form benötigt bei geschickter Programmierung, siehe Aufgabe 51, im Vergleich zu anderen Verfahren die wenigsten Rechenoperationen.

Ü 6.1.1 Ansatz mit verschiedenen Polynomtypen: Standard-Ansatz, Polynome in Newton-Form, diskrete Orthogonalpolynome

Aufgabe 48: Interpolation: Ansatz mit der Vandermonde-Matrix

Bestimmen Sie die Koeffizienten a, b, c, d des kubischen Interpolationspolynoms zum Datensatz aus Tabelle 1 mit dem Standard-Ansatz $p(x) = a + bx + cx^2 + dx^3$.

Überlegen Sie: Einsetzen der Datenpunkte führt auf ein Gleichungssystem mit der Matrix

$$\begin{bmatrix} 1 & 300 & 300^2 & 300^3 \\ 1 & 400 & 400^2 & 400^3 \\ 1 & 500 & 500^2 & 500^3 \\ 1 & 600 & 600^2 & 600^3 \end{bmatrix}.$$

Eine Matrix, deren Zeilen der Reihe nach die Potenzen von x-Werten enthalten, heißt *Vandermonde-Matrix*. Die gute Nachricht: Falls alle x-Werte verschieden sind, ist das Gleichungssystem eindeutig lösbar.
Die schlechte Nachricht: Diese Matrix kann sehr hohe Konditionszahl haben. Bei Polynomen höheren Grades entstehen gravierende Rundungsfehler. MATLAB's Befehl `polyfit` verwendet diesen Ansatz (mit all seinen Vor- und Nachteilen).

Aufgabe 49: Interpolation: Ansatz mit Polynomen in Newton-Form

Diese Aufgabe sollten Sie mit Stift auf Papier durcharbeiten. Sie erklärt Ihnen die wichtigen Eigenschaften des Newtonschen Interpolationsverfahrens.

Sie können das kubische Polynom auch in folgender Form ansetzen:

$$p(x) = a_0 + a_1(x - 300) + a_2(x - 300)(x - 400) + a_3(x - 300)(x - 400)(x - 500)$$

Setzen Sie hier die Wertepaare des (T, c_p)-Datensatzes aus Tabelle 1 ein. Schreiben Sie die zugehörige Koeffizientenmatrix an. Hier ist schon einmal ein Anfang gemacht:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 100 & 0 & 0 \\ & & \ddots & \\ 1 & 200 & & \\ 1 & 300 & \dots & \dots \end{bmatrix}.$$

Es treten hier praktischer Weise viele Nullen auf. Wie nennt man die spezielle Form einer solchen Matrix?

Bestimmen Sie die Koeffizienten a_0 bis a_3. Warum ist (mit Stift auf Papier) die Lösung des Systems wesentlich einfacher als die eines Vandermonde-Systems?

Der Rechenaufwand zur Lösung eines $n \times n$-Vandermonde-Gleichungssystems wächst mit $O(n^3)$. Mit welcher Potenz wächst der Rechenaufwand bei n Datenpunkten beim Newton-Ansatz?

Lassen Sie nun den letzten Term $a_3(x - 300)(x - 400)(x - 500)$ weg und betrachten Sie das quadratische Polynom

$$p(x) = a_0 + a_1(x - 300) + a_2(x - 300)(x - 400)$$

Ist dieses Polynom auch eine Art von Interpolationspolynom? Welche Daten lassen sich damit beschreiben?

Und wenn Sie auch noch den a_2-Term weglassen? Was beschreibt

$$p(x) = a_0 + a_1(x - 300) \quad ?$$

Sie wollen nun ein zusätzliches Wertepaar in die Interpolation einbeziehen:

T	cp
200	496.4

Fügen Sie dieses Wertepaar *am unteren Ende* der Datentabelle an und ergänzen Sie den Newton-Ansatz um einen weiteren Term

$$a_4(x-300)(x-400)(x-500)(x-600)$$

Was verändert sich an der Matrix des Gleichungssystems? was verändert sich am Ergebnis, also am Wert der Koeffizienten $a_0, a_1, \ldots$?

> Die einzelnen Terme im Newton-Interpolationspolynom beschreiben der Reihe nach lineare, quadratische, kubische… Interpolation zwischen zwei, drei, vier… Wertepaaren.
>
> Jedes zusätzliches Wertepaar ergänzt das Newton-Interpolationspolynom um einen weiteren Term; die anderen Terme bleiben unverändert.

Vergleichen Sie dazu den Polynom-Ansatz in Standard-Form $p(x) = a_0 + a_1x + a_2x^2 + \ldots$, wie ihn Aufgabe 48 verwendet. Da hätten das lineare, quadratischs, kubische… Interpolationspolynom jeweils völlig andere Koeffizienten.

Aufgabe 50: Interpolation: Ansatz mit diskreten Orthogonal-Polynomen

Auch bei dieser Aufgabe sollten Sie die Rechnungen mit Stift auf Papier durchführen. Sie sollen hier sehen: es gibt neben dem Lagrange- und dem Newton-Ansatz noch andere polynomiale Basisfunktionen.

Als dritten Ansatz wählen Sie nun

$$p(x) = a_0p_0 + a_1p_1 + a_2p_2 + a_3p_3$$

mit Ansatz-Polynomen

$$\begin{aligned}
p_0(x) &= 1,\\
p_1(x) &= \frac{1}{50}(x-450),\\
p_2(x) &= \frac{1}{10\,000}\big[(x-450)^2 - 12500\big],\\
p_3(x) &= \frac{1}{300\,000}(x-450)\big[(x-450)^2 - 20500\big]
\end{aligned}$$

Und wie kommt man auf diese Polynome? Die Regeln zum Konstruieren der Polynome $p_0, p_1, \ldots$ sind nicht besonders kompliziert, aber für den Lerneffekt in dieser Aufgabe nicht so wichtig. Nehmen Sie die Polynome als gegeben hin. Die Aufgabe will Sie dafür sensibilisieren:

- Neben der Standardbasis $1, x, x^2, x^3 \ldots$ gibt es noch andere, und oft günstigere Basis-Polynome, um den Raum aller Polynomfunktionen darzustellen.
- Orthogonalität ist eine wichtige Eigenschaft von Vektoren; diese Eigenschaft lässt sich auf Polynome übertragen.

Aufgabe 53 verwendet die Musterprogramme `orth_polyfit.m` und `orth_polyval.m` zur Konstruktion solcher Polynome, aber auch dort wird nicht näher darauf eingegangen, wie das funktioniert. Was Sie aber in den Musterprogrammen erkennen können: es sind wirklich nur wenige Code-Zeilen zur Konstruktion dieser Polynome nötig, nicht mehr als für Newton- oder Vandermonde-Ansatz.

Setzen Sie die Wertepaare des (T, c_p)-Datensatzes aus Tabelle 1 ein. Schreiben Sie die zugehörige Koeffizientenmatrix A an. Sie sieht harmlos und uninteressant aus:

$$A = \begin{bmatrix} 1 & -3 & 1 & \dots \\ 1 & -1 & & \\ 1 & & \ddots & \\ \vdots & & & \end{bmatrix}.$$

Sie hat aber eine wichtige Eigenschaft: $A^T \cdot A$ ist eine Diagonalmatrix. Anders gesagt: die Spaltenvektoren von A stehen aufeinander orthogonal. Damit erfüllt A fast die Definition einer Orthogonalmatrix – Welche Eigenschaft fehlt?

Die quasi-Orthogonalität von A vereinfacht die Lösung des Gleichungssystems für die Koeffizienten: Stellen Sie A auf; berechnen Sie $A^T \cdot A$ und $A^T\mathbf{b}$. Welcher einfache Schritt fehlt noch zur Lösung?

Die Ansatz-Polynome p_0, p_1, p_2, p_3 sind hier so speziell gewählt, dass ihre Werte für den x-Datenvektor (das sind genau die Spaltenvektoren in der A-Matrix) aufeinander orthogonal stehen. Man nennt Polynome mit so einer Eigenschaft *Orthogonalpolynome*. Speziell handelt es sich hier um *diskrete Orthogonalpolynome*, weil sich die Orthogonalität auf die diskreten Datenpunkte auf der x-Achse bezieht.

Falls Sie jetzt fragen: gibt es auch Polynome oder Funktionen, die nicht nur für einige diskrete x-Werte, sondern für alle x in einem bestimmtenm Bereich orthogonal sind? Ja, tatsächlich! Der Begriff „Orthogonalität“ lässt sich von Vektoren auf Funktionen übertragen.

Die wichtige Eigenschaft bei einem Ansatz mit Orthogonalpolynomem ist: Wenn Sie hier den letzten Term weglassen, erhalten Sie ein quadratisches Polynom. Anders als beim Newton-Ansatz interpoliert es zwar keine Datenpunkte mehr, aber es ist das beste Approximationspolynom (im kleinsten-Quadrate-Sinn) an die Daten.

Entsprechend erhalten Sie, wenn sie auch noch den vorletzten Term weglassen, mit $p(x) = a_0 p_0(x) + a_1 p_1(x)$ das beste lineare Polynom (die Ausgleichsgerade) an die Daten.

> Die einzelnen Terme in einem Ansatz mit Orthogonalpolynomen beschreiben der Reihe nach die bestmögliche lineare, quadratische, kubische… Approximation der Wertepaare. Letzlich, bei gleich viel Termen wie Wertepaaren, ergibt sich das Interpolationspolynom.

Aufgabe 51: Interpolation: Newtons Dividierte-Differenzen-Schema

Diese Aufgabe richtet sich an programmier-affine Leute, die wissen wollen, wie man Interpolationsverfahren möglichst rechengünstig implementiert. Solange Sie MATLAB als Rechenumgebung zur Verfügung haben, müssen Sie sich damit nicht befassen. Aber Sie lernen Sie hier clevere Programmier-Tricks, die sich anderswo einsetzen lassen.

Speziell geht es hier um effiziente Berechnung von Tabellen, wobei Zwischenergebnisse, sobald sie nicht mehr gebraucht werden, überschrieben werden, um Speicherplatz zu sparen.

Seite 66 im Skriptum zeigt eine Dreiecks-Tabelle für das kubische Interpolationspolynom nach Newton.

Bei vielen numerischen Verfahren sind Tabellen in Dreiecksform auszufüllen. Nicht alle Einträge werden zum Schluss gebraucht. Deswegen lässt sich der Rechengang geschickt so organisieren, dass immer nur eine Spalte der Tabelle mit gerade benötigten Werten gespeichert ist. Die neu berechneten Einträge überschreiben dann nicht mehr notwendige Zwischenergebnisse. Zum Schluss bleibt ein Vektor mit den benötigten Endergebnissen übrig.

Das folgende Code-Segment berechnet aus Datenvektoren x und y den Vektor dd als obere Schrägzeile des Dividierte-Differenzen-Schemas:

```
dd = y;
for i=2:n
    for k=n:-1:i
        dd(k) = (dd(k)-dd(k-1))/(x(k)-x(k-i+1));
    end
end
```

Implementieren Sie diesen Code als Funktion `dd=newtonpoly(x,y))` und überzeugen Sie sich für das durchgerechnete Beispiel im Skriptum (Seite 66) oder für die Daten der vorigen Aufgabe an einem Durchlauf im Debug-Modus, dass alle Zwischenresultate in dd berechnet werden und erst dann überschrieben werden, wenn sie für den weiteren Rechengang nicht mehr notwendig sind.

Die rechengünstige Auswertung eines Interpolationspolynoms in der Newton-Form an der Stelle z, wenn die Koeffizienten aus dem Dividierte-Differenzen-Schema auf einem Vektor dd und die x-Stützstellen auf x gespeichert sind, leistet eine Schleife der Form

```
y = dd(n);
for i=n-1:-1:1
    y = y.*(z-x(i)) + dd(i);
end
```

Implementieren und testen Sie eine Funktion `y = newtonval(x, dd, z)`, die für gegebene x-Datenpunkte und Dividierte-Differenzen-Koeffizienten dd das Polynom an der Stelle z auswertet.

Vergleichen Sie Anwendung der Funktionen `newtonpoly` und `newtonval` mit den auch MATLAB-Befehlen `polyfit` und `polyval`. MATLAB verwendet dabei den Vandermonde-Ansatz; für die Daten aus dem Skriptum sollte sich kein Unterschied ergeben.

Ü 6.1.2 Güte der Interpolation. Warnung vor hohem Grad. Runges Phänomen

Aufgabe 52: Runges Phänomen, Tschebyschoffs Stützstellen

Früher, in der präcomputerischen Ära, kam niemand leichtfertig auf die Idee, Interpolationspolynome zehnten oder zwanzigsten Grades zu verwenden. Heute reicht ein simpler MATLAB-Befehl, um Polynome hundertsten Grades zu verwenden. Aber, nur weil 's leicht geht, ist es auch vernünftig? Diese Aufgabe untersucht die Frage.

Berechnen Sie Interpolationspolynome für die Funktion

$$y = \frac{1}{1+x^2}, \qquad -5 \le x \le 5$$

an 5, 10, 15 und 20 äquidistanten Stützstellen. Am einfachsten verwenden Sie Matlabs `polyfit, polyval` oder den Code aus Aufgabe 51. Zeichnen Sie Funktion und Polynome. Interpretieren Sie die Graphik, experimentieren Sie auch noch mit höheren Polynomgraden.

Machen Sie sich in den Vorlesungs-Folien und in der Wikipedia zum Stichwort „Runge-Phänomen“ schlau[27] und beantworten Sie dann: Gilt bei polynomialer Interpolation automatisch: „je höher der Grad, desto besser die Näherung?“

Der Approximationssatz von Weierstraß besagt, dass sich jede stetige Funktion auf einem abgeschlossenen Intervall beliebig genau durch Polynome approximieren lässt. Warum werden dann in diesem Beispiel die Approximationen immer schlechter? Hat sich Weierstraß geirrt? – Natürlich nicht, Weierstraß war ja Mathematiker. Das Problem in diesem Fall sind die äquidistanten Stützstellen. Die meisten Funktionen (abgesehen von den völlig harmlosen sin, cos, exp-Funktionen) lassen sich nur extrem widerwillig durch äquidistante Stützstellen zwingen. Es ist natürlich möglich, aber die Funktion reagiert trotzig und weicht zwischen den Stützstellen mächtig vom vorgesehenen Verlauf ab. Siehe Runge-Phänomen

Besonders gegen den Rand des x-Bereiches hin kommt es zu heftigen Oszillationen. Deswegen ist es besser, zum Rand hin die x-Werte dichter zu legen. Besonders günstig im Intervall $[-5;\ 5]$ ist die Anordnung der Stützstellen gemäß der Formel

$$x_i = 5\cos\left(\frac{2i-1}{2n}\pi\right), \quad i = 1, \ldots, n \qquad \text{(Tschebyschow-Stützstellen).}$$

Interpolieren Sie die Funktion wieder mit 5, 10, 15 und 20 Stützstellen, die aber nicht äquidistant verteilt sind, sondern nach der obigen Formel. Experimentieren Sie auch noch mit höheren Polynomgraden und beantworten Sie dann: Gilt bei polynomialer Interpolation vielleicht doch: „je höher der Grad, desto bessere Näherung?“

Aufgabe 53: Approximation, sehr hoher Polynomgrad

Auch wenn bisher betont wurde, dass Polynome hohen Grades nur mit großer Vorsicht zu verwenden sind, gibt es Anwendungen (z.b. Bildverarbeitung, Daten entrauschen...), wo Approximationen mit hohem Grad sinnvoll sind. Hier untersuchen wir an einem einfachen Datensatz, welche numerischen Verfahren mit hohem Grad zurecht kommen. Die Datei `SatPressH2O.m` (Download von Übungs-Homepage) enthält reale Daten zum Sättigungsdampfdruck $p(T)$ von Wasser in Abhängigkeit von der Temperatur T.

Angenommen, sie wollen für diese Daten ein hochgenaues Approximationspolynom finden[28].

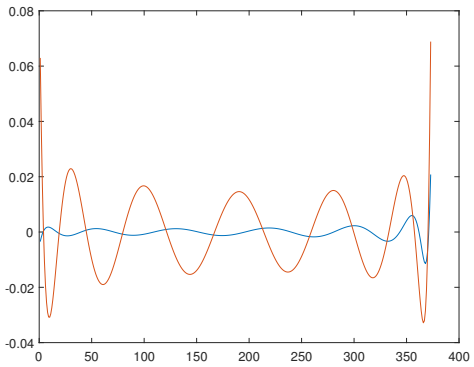
Vergleichen Sie drei Verfahren:

1. Klassischer Ansatz mit Vandermonde-Matrix und Aufstellen der Normalengleichungen
2. QR-Zerlegung der Vandermonde-Matrix – das machen die Matlab-Befehle `polyfit, polyval`
3. Ansatz mit diskreten Orthogonalpolynomen. Dazu gibt es Dateien `orth_polyfit.m` und `orth_polyval.m` zum Herunterladen; diese Dateien funktionieren analog zu `polyfit, polyval`, verwenden aber diskrete Orthogonalpolynome.

[27] Zur Aussprache: Carl Runge (1856–1927) war ein deutscher Mathematiker, sein Name reimt sich auf „Junge“, nicht auf den Musikstil „Grunge“.

[28] Für professionellen Einsatz sollte man die Daten noch geeignet umformen und skalieren; es macht auch praktisch wenig Sinn, Genauigkeit im Mikrobar- oder Nanocelsius-Bereich zu verlangen; niemand kann Temperaturen oder Drücke im praktischen Einsatz mit solcher Genauigkeit messen. Aber es geht ums Prinzip; die Ergebnisse sind typisch auch für andere Funktionen oder Datensätze.

Beginnen Sie mit Approximation durch ein Polynom 10-ten Grades und stellen Sie in einem Plot den Fehler in den Datenpunkten dar (Datenwert - Polynomwert). Erhöhen Sie schrittweise den Polynomgrad und stellen Sie fest, ob und wie weit der Fehler abnimmt. Je nach Verfahren lassen sich nur bis zu einem gewissen Grad sinnvoll Approximationen finden. Bis zu welchem Grad lässt sich die Methode der Normalengleichungen, die QR-Zerlegung, die Orthogonalpolynome numerisch stabil einsetzen?



Hier ist ein Beispiel-Plot: Er zeigt für Polynomgrad 12 in blau den Fehler des Approximationspolynoms aus der QR-Zerlegung (Matlab-Befehle `polyfit, polyval`), in orange den entsprechenden Fehler bei der Methode der Normalengleichungen. Er ist signifikant größer; das kommt nur von den Rundungsfehlern. Bei Grad 11 gibt es noch kaum Unterschied. Normalengleichungen können Sie also bei diesem Datensatz ab Grad 12 vergessen. Wie sieht es mit den anderen beiden Methoden bei höheren Graden aus?

Ü 6.2 Numerische Integration

Siehe die Folien der 6. und 7. Vorlesung. Im Skriptum unter Kapitel 8.

Das Romberg-Verfahren (Aufgabe56) ist nicht nur als genaues Verfahren zur numerischen Integration interessant, sondern illustriert ein allgemeines Prinzip: Berechne einen Näherungswert mehrmals mit unterschiedlicher Schrittweite $h_1, h_2, \ldots$, extrapoliere von diesen Daten auf $h = 0$. Diese allgemeine Idee heißt *Richardson-Extrapolation*. Ihre Anwendung auf die zusammengesetzte Trapezregel ist das Romberg-Verfahren.

Ü 6.2.1 Integration von Tabellenwerten

Wenn zu einer Funktion nur diskrete Datenpunkte vorliegen, kann daraus ihr Integral näherungsweise mit verschiedenen numerischen Quadraturformeln berechnet werden.

Hier sind nochmals die Tabellenwerte der spezifische Wärmekapazität von kohlenstoffarmem Stahl gegeben.

T	cp
0	460.8
100	471.1
200	496.4
300	537.0
400	593.3
500	666.8
600	760.8

Spezifische Wärmekapazität von kohlenstoffarmem Stahl in J/kg K für Temperaturen zwischen 0 und 600C

Aufgabe 54: Integral der spezifischen Wärmekapazität

Das Integral der spezifischen Wärmekapazität ist die Enthalpie. Berechnen Sie für die gegebenen Daten das Integral von $T = 0$ bis $T = 600$

- mit der Simpson-Regel (3 Stützstellen $T = 0, 300, 600$)
- mit der 3/8-Regel (4 Stützstellen $T = 0, 200, 400, 600$)
- mit der zusammengesetzten Trapezregel ($h = 100$)
- mit der zusammengesetzten Simpson-Regel ($h = 100$)

Hinweis: In MATLAB lässt sich die Multiplikation von Funktionswerten mit Gewichtsfaktoren elegant als Skalarprodukt zweier Vektoren schreiben. Am Beispiel der 3/8-Regel:

```
b=600;
a=0;
f= [460.8   496.4   593.3   760.8];
w=[1 3 3 1]/8;
int38=(b-a)*f*w'
```

Ü 6.2.2 Numerische Integration von Funktionen

Wenn eine Funktion nicht in geschlossener Form integrierbar ist, oder der Rechenausdruck zu kompliziert wird, ist numerische Integration (auch: *numerische Quadratur*) möglich. Der MATLAB-Befehl `q = quad(fun,a,b)` berechnet das Integral von f in den Grenzen von a nach b mit einer adaptiven Simpson-Regel. Dabei kann f *inline* gegeben sein,

```
>> q=quad('1./x',1,2)
q =
    0.6931
>>
```

Achtung, punktweise Vektor-Rechenoperationen `./`, `.^` usw. verwenden

auch als anonymous function oder als Funktions-Henkel, genauso wie beim Aufruf von `fzero()`.

Aufgabe 55: Die Agnesi-Kurve

Am 16. Mai jährt sich zum 305. mal der Geburtstag von Maria Gaetana Agnesi, einer altösterreichischen Mathematikerin, die Kurven vom Typ

$$y = \frac{a^3}{x^2 + a^2}$$

untersucht hat. Ihr zu Ehren sollen Sie diese Kurve numerisch integrieren. Berechnen Sie numerisch das Integral

$$\int_{-1}^{1} \frac{1}{x^2+1} dx \qquad \text{(exakter Wert: } \frac{\pi}{2}\text{)}$$

(a) mit der zusammengesetzten Trapezregel und drei verschiedenen Schrittweiten: $h = \frac{4}{10}, \frac{2}{10}, \frac{1}{10}$. Vergleichen Sie mit dem exakten Wert $\frac{\pi}{2}$ und geben Sie an: Bei halber Schrittweite reduziert sich der Fehler (annähernd) um einen Faktor ...

(b) mit MATLABs `quad`-Befehl

Aufgabe 56: Die Agnesi-Kurve mit Romberg-Integration

Romberg-Verfahren: Nennen Sie die drei Ergebnissen von Punkt a der vorigen Aufgabe (Trapezregel mit jeweils halbierter Schrittweite) $T_{1,1}, T_{1,2}, T_{1,3}$, Berechnen Sie ein Dreiecks-Schema der Form

$$\begin{array}{lll} T_{1,1} & & \\ & T_{2,2} & \\ T_{1,2} & & T_{3,3} \\ & T_{2,3} & \\ T_{1,3} & & \end{array}$$

nach der Vorschrift

$$T_{2,2} = \frac{4T_{1,2} - T_{1,1}}{3} \quad , \quad T_{2,3} = \frac{4T_{1,3} - T_{1,2}}{3} \text{ und } , \quad T_{3,3} = \frac{16T_{2,3} - T_{2,2}}{15}$$

Die Resultate werden zunehmend genauer. Vergleichen Sie mit dem exakten Resultat!

Und jetzt der allgemeine Fall (Bonus-Aufgabe): die Formel des Romberg-Verfahrens bei k Ergebnissen $T_{1,1}, T_{1,2}, \ldots T_{1,k}$ aus der Trapezregel mit jeweils halbierter Schrittweite lautet:

$$T_{i+1,j} = \frac{4^i T_{i,j} - T_{i,j-1}}{4^i - 1}$$

Dabei bezieht sich der Index i auf die Spalten $1, 2, \ldots k$ des Dreiecksschemas, Index j bezeichnet die Aufwärts-Schrägzeilen.

$$\begin{array}{llllll} T_{1,1} & & & & & \\ & T_{2,2} & & & & \\ T_{1,2} & & T_{3,3} & & & \\ & T_{2,3} & \vdots & \ldots & T_{k,k} & \\ T_{1,3} & \vdots & T_{3,k} & & & \\ \vdots & T_{2,k} & & & & \\ T_{1,k} & & & & & \end{array}$$

Sie können dieses Dreiecksschema Speicherplatz sparend auf ähnliche Weise berechnen, wie die dividierten Differenzen des Newton-Schemas in Aufgabe 51 – wenn Sie das schaffen, wäre das dann noch mal ein Extra-Bonus!

Die $T_{i,j}$ Werte nähern mit zunehmender Genauigkeit; wenn sich Werte nicht mehr ändern, (innerhalb einer Fehlerschranke), kann man davon ausgehen, ein hinreichend genaues Resultat erreicht zu haben. Adaptive Quadraturformeln arbeiten im Wesentlichen nach diesem Prinzip.

Ü 6.3 Analytische Integration von Funktionen

Bonus-Material für Interessierte. MATLAB kann symbolisch nicht nur differenzieren, sondern auch integrieren. Das ist gut zu wissen, und wir laden Sie herzlich ein, den Abschnitt durchzuarbeiten.

Während es zum Differenzieren fixe Regeln gibt, die schrittweise zum Ergebnis führen, ist analytische (man sagt auch: symbolische) Integration wesentlich komplizierter. Nur für recht einfache Funktionen gibt es Integrationsregeln; darüber hinaus findet man eine Sammlung von Tricks und „Kochrezepten“. Oft läßt sich nicht von vornherein absehen, ob eine gegebene Funktion eine Stammfunktion hat und welche Methode ein Ergebnis liefern wird.

Früher gab es umfangreiche Integraltafeln zum Nachschlagen, jetzt übernehmen Computeralgebrasysteme diese Aufgabe. Die sind inzwischen sehr leistungsfähig und können für viele Funktionstypen Integrale berechnen. Der zuständige Befehl in MATLABs *symbolic math toolbox* lautet `int( )`.

Zum Aufwärmen:

```
>> syms x
```

Definieren Sie x als symbolische Variable

Damit können Sie Ihre Mathematik-Kenntnisse auffrischen: Wie lautet $\int x^2\,dx$?

```
>> int(x^2)
ans =
1/3*x^3
```

Gibt MATLAB die richtige Antwort? Lassen Sie sich dazu eine Geschichte erzählen...

> Zwei Mathematiker sitzen beim Bier und klagen über das allgemein recht spärliche mathematische Grundlagenwissen. Dann muss der erste Mathematiker aufs Klo, das nützt der zweite aus und ruft die Kellnerin. Er erklärt ihr, in ein paar Minuten, wenn sein Freund wieder zurück sei, werde er sie nochmals zum Tisch bitten und etwas fragen. Sie müsse einfach nur antworten: „Ein Drittel x der Dritten". Sie fragt nach: „Ein Tritt – Elixta tritt ihn??" Er wiederholt geduldig: „Ein Drittel x der Dritten". Darauf sie: „Ein Driddelix, der tritt 'n?" „Ja, genau so", seufzt er. Gut, sie ist einverstanden und murmelt nun immerfort in sich hinein: „Ein Driddelix, der tritt 'n..."
>
> Der erste Mathematiker kommt zurück und der zweite schlägt vor: „Was wetten wir, die meisten Leute haben sehr wohl etwas Ahnung von Mathematik. Ich frage die blonde Kellnerin dort nach einem Integral". Lachend nimmt der erste die Wette an. Der zweite ruft also die Kellnerin und fragt: „Was ist das Integral von x Quadrat?" Die Kellnerin sagt: „Ein Drittel x der Dritten", dreht sich um und ruft im Weggehen über die Schulter hin noch zurück, „plus C!".

Versuchen Sie weitere Integrale:

```
>> syms x n t
```

Definieren Sie noch einige symbolische Variable

```
>> int(x^n)
ans =
piecewise([n = -1, log(x)], [n <> -1, x^(n + 1)/(n + 1)])
>>
```

Das sieht kompliziert aus, ist aber korrekterweise MATLABS Interpretation der Regel

$$\int x^n\,dx = \begin{cases} \log x & (n=-1) \\ \frac{x^{n+1}}{n+1} & (n\neq -1) \end{cases} + C$$

```
>> int(sin(t))
ans =
-cos(t)
>>
```

Auch die gängien Funktionen lassen sich integrieren...

```
>> int(exp(-x^2))
ans =
(pi^(1/2)*erf(x))/2
>>
```

...obwohl es schnell kompliziert werden kann. Hier ist ein Integral nicht durch elementare Funktionen darstellbar. Es gibt aber die spezielle Funktion erf (die eigens für Integrale dieses Typs erfunden wurde).

```
>> int(cos(n*t))
ans =
1/n*sin(n*t)
>>
```

Hat MATLAB hier nach n oder nach t integriert?

MATLAB versucht (ähnlich wie beim Differenzieren) die Integrationsvariable zu erraten. Es nimmt an, Buchstaben am Ende des Alphabets seien Variable, Buchstaben weiter vorne eher Parameter und Konstante. Deswegen interpretiert es `int(cos(n*t))` als $\int \cos(nt)\,dt$ und nicht als $\int \cos(nt)\,dn$. Im Zweifelsfall akzeptiert der `int()` Befehl ein zweites Argument:

```
>> int(log(x)*cos(t),x)
ans =
x*cos(t)*(log(x) - 1)
>>
```

Das ist $\int \log x \cos t\,dx$

```
>>int(log(x)*cos(t),t)
ans =
log(x)*sin(t)
>>
```

Das ist $\int \log x \cos t\,dt$

Weitere Möglichkeiten: `int(S,a,b)` gibt das bestimmte Integral von S in den Grenzen von a bis b, `int(S,v,a,b)` legt zusätzlich die Integrationsvariable v fest.

Aufgabe 57: Symbolische Integration

Berechnen Sie symbolisch in MATLAB

$$\int \frac{1}{1+x^2}\,dx \qquad \int \sqrt{1-ax^2}\,dx$$

$$\int e^{-tx^2}\,dt \qquad \int_1^2 \frac{1}{x}\,dx$$