Chapter 3.2 - Numerical Integration

13 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=7KHWDMNJERS
https://www.youtube.com/watch?v=7khWDmNjERs

SUMMARY

Numerical integration is a more stable and smoother operation compared to differentiation, especially when dealing with noisy data. This summary discusses the fundamental concepts of numerical integration, including various methods like the trapezoidal rule and Simpson's rule, which help approximate integrals while managing errors effectively.

- *Numerical integration is more stable than differentiation and is used to compute areas under curves.*
- *The integral is defined as a limit process, approximating areas using small rectangles or triangles.*
- *Common methods for numerical integration include the trapezoidal rule and Simpson's rule, which improve accuracy by using different shapes (rectangles vs. parabolas).*
- *The trapezoidal rule approximates the area by averaging the heights of two points, while Simpson's rule uses three points to fit a parabola.*
- *Error estimates for these methods can be derived using Taylor series expansions, with Simpson's rule providing a better accuracy (H^4) compared to the trapezoidal rule (H^2).*
- *Composite rules can be used to efficiently compute integrals over larger intervals by combining results from smaller segments.*
- *Recursive refinement allows for improving the accuracy of integral calculations without repeating all computations, leveraging previously calculated values.*
- *Implementing these rules in programming (e.g., Python) can be straightforward, allowing for quick and efficient numerical integration.*

so in addition to differentiation we want to think about integration now integration as a numerical scheme is much more stable than differentiation in fact it's a smoothing operation right so differentiation which is trying to find slopes can be very difficult to do when you have noisy data we'll be talking about that in a couple Le from now but numerical integration is actually a fairly stable scheme to compute areas under the curve and so let's talk about what numerical

integration is let's go back again to the definition of an integral the definition of an integral is through a limited process so we typically represent this integral from A to B of X DX as a limit of H goes to zero of the function f of XJ * H and what H is is essentially the width of a little triangle so here here's the here's the standard picture we learned in calculus right is like you add up the areas of

all these little rectangles where you evaluate the function at points X1 X2 so which you have the values F1 ff2 F3 and so forth which are the heights of the triangles and the width of these triangles are going to be H so I add these up and I'm taking the limit as H goes to zero in other words these triangles become infinitely thin right

and then as I make them infinitely thin I actually get this limiting process which we call the

integral okay so that's the idea behind this and just like differentiation we're not going to be able to take this limit what we're going to do when we do computations on the computer this H is not going to go to zero H is going to be small and the question comes how well do I approximate an integral then and what's the error I get in doing so so those are the things we're actually after is not only the approximation but

how accurate is the approximation so again the definition of a girl would take some curve like this function f break it up into a bunch of basically structures that we want to add up each with with h and we can approximate this in various ways we can use trapezoid sorry trapezoid rule or we can use re simple rectangles so there are different formulas for this and we can actually even start to approximate this function in more

interesting ways now the easiest way to approximate this is through what's called a quadrature rule and a quadrature rule rule basically defines how we want to specifically add up these rectangles or we can make them even more exotic than rectangles if we like so this is the generic formula it's a sum of all those of all structures with weights WK but evaluated the function f at different points X of K so right we're going to actually Define

we're going to evaluate this function at very specific points and the question is what weight do we want to give to it which is determined by WK so this is generically called the quadrature rule and as you could imagine depending upon how we pick the weights we're going to be able to control our error and accuracy and this is exactly what's going to happen in here for us so let's talk about some of the standard things that you might do

the simplest thing to do is what we learned in calculus which is to do something like a trapezoid rule which we'll talk about a minute and in doing so we're going to get an error so when we approximate this derivative we're going to get the approximation is going to be the quadrature plus some air and so the simplest thing is the trapezoid rule which is you take this function f ofx DX and you approximate it for instance from X notot to X X1 you

just simply take the average height from between F and F1 the average height is f F1 over2 * its width H so what you're doing is taking a rectangle of the average height between two neighboring points adding those all up and by the way if you do a tailor series expansion around these points you can actually show that here's the error again so then tailor series expansion again plays this key role in defining our error and so when you do

this you get something that's H cub so this is your err but now notice this is H in front of this H cubed in front of this so this is H squar smaller than this and so we typically think of trapezoid rule as an as a h squ accurate scheme you can improve on this and one of this the most important improvements that we have that's a very simple generalization it was called simpsons's Rule and what Simpson's rule does it

adds not just a single not just it doesn't just simp simply say two points and draws a rectangle between them what it does is it takes three points and fits a parabola to the three points and then finds the area under the curve

there and in doing this this is the formula you have the width H over3 the point fnot 4 F1 4 F2 these are three points and the error is H5 so what you did is you pushed your error out

from H squ all the way to H4 again just like when we did numerical differentiation we had Delta t^ s accurate schemes and delta T to the 4th accurate schemes we want the more accurate schemes of possible Simpsons Ru is a very simple generalization that improves our accuracy significantly there's also what's called Simpsons 38 rule which uses four points or something like Bull's rule that get you all the way to you know an h66 accurate scheme and what it does

just like in you do a numerical differentiation the more points you use the more you can push down the airor and you can find this and you can show this all through Taylor series expansions so let me just show you these formulas in sort of a graphical way so essentially when you do trapezoid rule it's like basically saying look I want to just fit a trapezoid here which is the average height of this times the width Simpson

rule basically fits a parabola through these three points and in fitting that Parabola through these three points point you improve your error estimate significantly here's Simpsons 38 Rule and here is Bull's rule so notice you use more neighbors and you can often improve your accuracy so these are known as the Newton's coats formulas for integration so they're very standard to use Simple to program and so let's talk about actually programming them and also for each of

those rules right that was for a single cell but really what I need to do is now approximate over the entire A to B right so from A to B you know this is now a sum over all those triangles or a sum over all those par parabolas that are sitting there across the entire domain so let's just consider for a moment the trapezoidal rule which is this expression here so if I write it out here's my first trapezoid h over2 f

plus F1 second trapezoid H over2 F1 + F2 and so forth all the way through but notice I have an F1 here and an F1 here I can combine these so in fact if I were to take these I have an F1 here F1 here I'm an F2 here there's an F2 in the next so if I were to just implement this I would keep summing the same number twice but I could instead ahead of time just realize

well I have one I have a H2 F1 here H2 F2 there's a total then of two of them and so I have now a composite rule which is this is now collapsing this by combining all the terms that I have in common and so doing what I have is this final formula which is a composite rule which is H over 2 the two end points plus two times all the interior points so this is you know this saves this

computation is about twice as fast and remember with Computing we are actually going for speed so in a lot of modern Computing when we're doing this you might not even feel this at all you don't really care about how fast it is because the computers are so fast today but the fact is in general as a philos philosophical point you always want to be Computing in the most efficient fastest way possible so these composite rules are important because

what they're doing is allowing you to compute the trapezoidal rule Simpsons rule all in a in a in a rapid fashion so let's write a little code for this so here's a little python script so for us to basically compute both the trapezoid Rule and and Simpsons rule it's very simple right so again you can all this codes available on the GitHub so we're going to just make an X range which is MPA range from Zer to 2 pi steps of H and I've defined

H here I'm going to take the two Pi the pi two Pi domain and chop it up into 50 points so this going to be my determine my H so I could have more than 50 100 points 200 points whatever I want to do I can control the error by doing this and I'm going to integrate cosine which I know is s okay so in fact once you integrate this you can plot it all to see how well

you did right so here's my area under the curve and p z first and so I start a running computation so this is the cumulative integral as I walk across so I'm going to go from for J and range 0 to n minus one and here's just my trapezoidal rule it's h over 2 and the first point plus the second average of the points so you take the height of H knot H1 over two times the

width that's it and you add it up so this is the accumulative area as you go along and so that's as simple as it gets right that's the trapezoidal rule just using its neighbors Simpsons rule is pretty similar it's just as easy but now here what you have is The Simpsons rule which is the first point four of the next point one of the next Point divide by 3 * H it's about as easy as it gets so right

these are very simple rules and in fact if you wanted to you could write them one time ahead of time and then you just have them in a library defined as a function or you can even call on python has code that will do Simpsons rule directly for integration but I just wanted to show those curves there because they're they're important that you know how to at least program these basic structures so let's talk about one

last thing which is recursive refinement what we know is that H determines the accuracy now we may for instance decide that we produced an integral and we want to in fact refine our integral computation and make it even more accurate now suppose you've already done the integral so like for instance here now I'm taking step sizes of 2H so I'm going to go from X knot to X2 which is that width is 2H so it's using

f plus F2 as my basic trapezoid that I have so I could use this as a as a as a quadrature rule and use a sum where the 2H is the width of of my trapezoids and in doing this I end up with the following formula so right so I just took a bigger width part of the reason you might take a bigger width it's a faster computation to compute this integral and maybe it's the accuracy is good for what you need

so here's your composite rule written out for this but suppose you weren't happy with it then what you could do is say well actually I want more accuracy now you could come back and recompute all of this or realize that most of the computation you already have in this in this computation already notice you have the sum all you really need is to realize that what I'm missing is the contribution of the points cuz I have

the points at 0 2 4 6 8 I'm missing the contributions from the point at 1 3 5 and so forth and so what you can do is a recursive refinement by realizing that the quadrat tural calculation with h as your width is equal to half of the quadrature with width 2H plus this extra expression here and this is really actually not very hard to work out and so all you have to do is you take your

old computation which is the quadrature a fatter discretization and now you just add H times the missing points and now what you've done is you've improved your computation and you've recycled we love to do this in Seattle so what you've recycled here is these computations so you don't have to do them again all you do is add these new ones so that's a smart way to do this to improve your refinement until you get get a better

answer and it's a very simple thing to do without having to do all the work over again so these are the kind of things we think about with integrals and I think in general you want to think about differentiation and integration as two really important components of what you want to be doing overall in your scientific Computing framework