Lecture Content Summaries for MATH3201 (from week 4)

Travis Mitchell

Updated: 21 March, 2019

1 Week 4

1.1 Recap of the first three weeks

Started by looking at *Taylor series* approximations for a function value $f(x)$ near a known point $f(a)$:

$$f(x) \approx f(a) + (x-a)f'(a) + \frac{1}{2}(x-a)^2 f''(a) + \frac{1}{3!}(x-a)^3 f^{(3)} + \cdots + \frac{1}{n!}(x-a)^n f^{(n)}(a) \tag{1}$$

This we refer to as a Taylor approximation up to order n. This is how our expansion of $f(x)$ about $f(a)$ looks for a uni-variable function, but what about multivariate systems? An example for a two dimensional system would mean formulating our expansion of $f(x,y)$ about a point (a,b) which looks like:

$$\begin{aligned} f(x,y) =& f(a,b) + (x-a)f_x(a,b) + (y-b)f_y(a,b) + \\ & \frac{1}{2!}\left[(x-a)^2 f_{xx}(a,b) + 2(x-a)(y-b)f_{xy}(a,b) + (y-b)^2 f_{yy}(a,b)\right] + \text{H.o.T} \end{aligned} \tag{2}$$

This we can then write in a more general setting for a system of N variables by using our gradient and Hessian operators:

$$f(\mathbf{x}) = f(\mathbf{a}) + (\mathbf{x}-\mathbf{a})^T \nabla f(\mathbf{a}) + \frac{1}{2}(\mathbf{x}-\mathbf{a})^T H(\mathbf{a})(\mathbf{x}-\mathbf{a}) + \text{H.o.T} \tag{3}$$

With these expressions, we now have the base level of knowledge required to start looking at approximating derivatives. Initially, we looked at three approximations for the first derivative of a function $f(x)$ as well as an approximation for the second order derivative. The expressions for these are given in Table 1.

Table 1: Expressions and the associated order of error for first and second order derivative approximations.

Approximation	Expression	Order Error
1st order forward difference	$f'(x) \approx \frac{f(x+h)-f(x)}{h}$	$O(h)$
1st order backward difference	$f'(x) \approx \frac{f(x)-f(x-h)}{h}$	$O(h)$
1st order centred difference	$f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$	$O(h^2)$
2nd order difference	$f''(x) \approx \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$	$O(h^2)$

From here, we investigated possible techniques for finding higher order derivatives and the value of leading order error terms. In particular an example using the **Method of Undetermined Coefficients** was given in W2L1. Additionally, the natural extension for partial derivatives was given, e.g.

$$f_x(x,y) \approx \frac{f(x+h,y)-f(x-h,y)}{2h}. \tag{4}$$

The next topic introduced was *root finding* in the real domain, $\mathscr{R}^n$. Namely, this means we are wanting to solve the expression $f(x) = 0$, for which Newton's method provides a nice follow on from Taylor series expansions. In order to use Newton's method, we first make an initial guess, x_0, and then iterate using,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{5}$$

Here it is clear that the derivative of the function cannot be zero at x_n, and we also need to compute the derivative (note that this can be done with the approximations discussed if an analytical form cannot be found). When considering this method, be sure to understand its limitations and times when it can go unstable!

The method can be extended to solve algebraic systems as well by making use of the Jacobian, $\mathscr{J}(\mathbf{x})$ and the update rule,

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathscr{J}^{-1}(\mathbf{x}_n) \cdot F(\mathbf{x}_n), \tag{6}$$

where $F(\mathbf{x})$ is the vector of functions, $F(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_n(\mathbf{x}))^T$.

The issue with this method comes from the computational expense of computing the inverse of the Jacobian.

A further extension of Newton's method can be used to solve for maxima and minima of functions, namely we would be solving for $\nabla f = 0$, rather than $f(\mathbf{x}) = 0$. For this, the Newton update would become:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \left[\mathscr{H}(f(\mathbf{x}_n))\right]^{-1} \cdot f(\mathbf{x}_n), \tag{7}$$

where $\mathscr{H}$ is the Hessian matrix. To then classify the result, we use the eigenvalues of the Hessian matrix,

$$\text{if} \begin{cases} \lambda_i > 0 \quad \forall \quad i, \implies \text{ minimum} \\ \lambda_i < 0 \quad \forall \quad i, \implies \text{ maximum} \\ \lambda_i \neq 0 \quad \forall \quad i \text{ but signs are not consistant,} \implies \text{ saddle} \\ \lambda_i = 0 \quad \forall \quad i, \implies \text{ inconclusive.} \end{cases} \tag{8}$$

The method of steepest descent provides another means for finding local minima of a function $f(x)$. In this approach, we start from an initial guess, $\mathbf{x}_0$, and move in the direction of steepest descent (as the name may suggest!). The update rule for this method is given as:

$$\mathbf{x}_{n+1} = \mathbf{x} - \lambda \nabla f \tag{9}$$

Here we compute $\mathbf{x}_{n+1} = \mathbf{x}_{n+1}(\lambda)$ and then substitute this into $f^* = f(\mathbf{x}_{n+1}(\lambda))$. From here, we solve λ for,

$$\frac{d}{d\lambda} f^* = 0, \tag{10}$$

and use this to solve for $\mathbf{x}_{n+1}$.

1.2 With that out of the way, Week 4 content - *Integration*!

The first technique for numerical integration that we introduce is the *Trapezoidal rule*, this consists of dividing the domain up into Trapezoids and then summing the area of these,

$$\int_a^b f(x)dx \approx h\left[0.5(f_0 + f_N) + \sum_{i=1}^{N-1} f_i\right]. \tag{11}$$

In order to analyse the error, we can look at a single trapezoid and use a Taylor expansion then compare this to the exact integral. From this, we obtain an error of $O(h^3)$, but be a little careful here as this is the associated error for a single trapezoid. Summing these over the entire domain gives the global error as $O(h^2)$.

To improve on the accuracy of the Trapezoidal rule, we can also use a local quadratic approximation for the function and this is called *Simpson's rule*. This means we need three points for the approximation, so locally becomes:

$$\int_{x_i}^{x_{i+2}} f(x)dx \approx \frac{h}{3}(f_i + 4f_{i+1} + f_{i+2}) + O(h^5). \tag{12}$$

Then again, summing these sub intervals over the full domain increases the order of error globally to, $O(h^4)$.

The question now comes, is there a smart way to choose the step size h? This is where *adaptive quadrature* comes into play. This technique allows us to have the computer determine whether finer resolution is necessary at certain points in our domain. For example, with the Trapezoidal rule, we can compare the integral for a single trapezoid with that of the same trapezoid computed with 2 half intervals. The error here can effectively be compared to a tolerance to determine if sufficient resolution has been used.

2 Week 5

2.1 Adaptive integration

If may be common that we have a function or data set that we need to integrate, but there could be an 'uninteresting' region as well as regions of rapid change - using a constant interval size for the full domain for such a circumstance can be computationally excessive. Additionally, it can be simpler and more convenient for a user to input in a tolerance they want to achieve with the integration rather than specifying an interval number.

Adaptive trapezoidal rule

Compare integration value for a region using a single trapezoid, $S_{[x_i,x_i i+1]}$, compared to two, $S_{[x_i,u_i]}+S_{[u_i,x_{i+1}]}$. From this we find that:

$$\text{error}(S_{[x_i,x_i i+1]} - S_{[x_i,u_i]} - S_{[u_i,x_{i+1}]}) \approx 3 \times \text{error}(S_{[x_i,u_i]} + S_{[u_i,x_{i+1}]}) \tag{13}$$

Therefore, we can take our criterion for convergence as:

$$|S_{[x_i,x_i i+1]} - S_{[x_i,u_i]} - S_{[u_i,x_{i+1}]}| < 3 \times tol \tag{14}$$

Note that to achieve a specified tolerance, we need to account for the expected error when we half an interval. Therefore, when the algorithm fails the above criterion and we half our interval the required tolerance must also be halved.

Adaptive Simpson's rule

Compare integration value for a region using a single trapezoid, $S_{[x_i,x_i i+1]}$, compared to two, $S_{[x_i,u_i]}+S_{[u_i,x_{i+1}]}$. From this we find that:

$$\text{error}(S_{[x_i,x_i i+1]} - S_{[x_i,u_i]} - S_{[u_i,x_{i+1}]}) \approx 15 \times \text{error}(S_{[x_i,u_i]} + S_{[u_i,x_{i+1}]}), \tag{15}$$

but as a conservative rule of thumb, 15 is replaced by 10.

2.2 Ordinal differential equations

Wanting to work out numerical ways to solve problems of the type:

$$\frac{dy}{dt} = f(t,y) \tag{16}$$

given the initial condition, $y(t_0) = y_0$.

A simple way we can approach this is to expand about our initial condition with a Taylor series:

$$\begin{aligned} y(t_0+h) &= y(t_0) + hy'(t_0) + O(h^2) && (17)\\ &\approx y(t_0) + hf(t_0, y(t_0)) && (18)\\ &\vdots && (19)\\ \implies y_{n+1} &= y_n + hf(t_n, y_n) && (20) \end{aligned}$$

Therefore, given our initial condition and the derivative we are able to progress the function $y(t)$ through time. This update rule is known as **Euler's method**, and in practice is not used due to the high order of error which adds up cumulatively as we move further in time. One possible option is to Taylor expand to higher orders, but in practice is also not done - however, from this thinking leads the family of Runge-Kutta methods for solving ODEs.

Runge-Kutta methods

To obtain these, we often perform an expansion with Taylor series and then look at using the chain rule to breakdown higher order derivatives. Some common solvers that we will use in this class include:

- Runge-Kutta method of order 2 (also referred to as the midpoint rule)

$$y_{n+1} = y_n + hf\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} f(t_n, y_n)\right) + O(h^3) \tag{21}$$

- Runge-Kutta method of order 4

$$\begin{aligned} y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 &= f(t_n, y_n) \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_1\right) \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_2\right) \\ k_4 &= f(t_n + h, y_n + hk_3) \end{aligned} \tag{22}$$

The general method for deriving Runge-Kutta methods is given in Wk6 L1.

3 Week 6

3.1 Stiff problems - implicit methods

Here, we formulated the integral over a section of the ODE by taking the height of a rectangle created at $y(t_{n+1})$, rather than t_n which led to the forwards Euler method. By doing this, we came up with the implicit Euler method;

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) \tag{23}$$

Taking this approach leads us to a system of equations to solve, for which something like Newton's method can be used to solve for the solution. This requires more computation than the forward Euler method, but has improved stability properties making it essential for certain problems.
So what makes a problem stiff?

- "If a numerical method with a finite region of absolute stability applied to a system with any initial condition is forced to use, in a certain interval of integration, a step length which is excessively small in relation to the smoothness of the exact solution in that interval, then the system is said to be stiff in that interval;"
- Stiffness occurs when stability requirements, rather than those of accuracy, constrain the step size;
- Stiffness occurs when some components of the solution decay much more rapidly than others.

3.2 Systems of ODEs

As most higher order ODEs can be written as a system of first order ODEs, we can use previously analysed methods to solve these. For example we can solve using Euler methods, but now we have vectors,

$$\mathbf{y}^{n+1} = \mathbf{y}^n + h\mathbf{f}(\mathbf{y}^n, t^n), \tag{24}$$

where $\mathbf{y} = \{y_1, y_2, ..., y_n\}$, and $\mathbf{f} = \{f_1, f_2, ..., f_n\}$.

This can also be applied with higher order methods, such as the RK2 and RK4, previously shown for 1 variable.

3.3 Boundary value problems for ODEs

Shooting method

Here, rather than being given initial values allowing us to predict forward in time, here we are now given a start and end point that our solution must comply with, in general we can write this as,

$$\frac{d^2y}{dx^2} = f\left(x, y(x), \frac{dy}{dx}(x)\right), \quad y(a) = y_a \text{ and } y(b) = y_b. \tag{25}$$

The methodology is to first rewrite the second order problem as a system of first order ODEs. We then guess, for example, the initial derivative and use this to 'shoot' at the final solution with an ODE solver. This process can be written into its own function, returning an error of our shoot compared to the boundary value, and hence a root finding method can be used to solve.