MATH 405 Summary

Reese Critchlow

Introductory Concepts

Discretization of Functions: Given some function $f : \mathbb{R} \to \mathbb{R}$ in some specified domain $D = [a, b]$, we can *discretize* the function over the domain D over N points:

$$f \approx \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_N) \end{bmatrix}$$

Bisection Search/Scalar Problem: Say for some $f : \mathbb{R} \to \mathbb{R}$, we wish to find the point α where $f(\alpha) = 0$.

Hence, we can perform a bisection search on a "root bracket", which is define by some $[a, b]$ such that $f(a)f(b) < 0$, provided that f is a continuous function.

Bisection Search Algorithm:

```
while not |b - a| < eps:
    m = (a - b) / 2
    if f(m) * f(b) < 0:
        a = m
    else:
        b = m
```

The analysis of this function supports that $m_k \to \alpha$ as $k \to \infty$.

Linear Convergence: We define a linearly convergent method to obey the following rule:

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|} \leq \gamma \leq 1$$

If $\gamma = 0$, this would be called "super-linear".

Quadratic Convergence:

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^2} \leq k$$

Newton's Method: From a second order Taylor expansion, we can find a quadratically converging method, *Newton's Method* for root finding. It is as follows:

$$x_{k+1} = x_k - \frac{1}{f'(x_k)} f(x_k)$$

We can also apply this to systems of equations or vector-valued functions:

For some $\vec{x_k}$, solve $J(\vec{x_k})\vec{\delta} = -f(\vec{x_k})$, then:

$$\vec{x}_{k+1} = \vec{x}_k + \vec{\delta}$$

Where $J(\vec{x})$ is the *Jacobian*:

$$\begin{bmatrix} (f_1)_{x_1} & \cdots & (f_1)_{x_d} \\ \vdots & \ddots & \vdots \\ (f_n)_{x_1} & \cdots & (f_n)_{x_d} \end{bmatrix}$$

Lipschitz Continuity: We say that a function $f : D \to R$ is *Lipschitz Continuous* with value L, if it holds that:

$$\forall u,\, v \in D,\ \ |f(u) - f(v)| \leq L|u - v|.$$

Convergence of Newton's Method: If $f(\vec{\alpha}) = \vec{0}$ and $J(\vec{x})$ is invertible and Lipschitz continuous in a neighbourhood of $\vec{\alpha}$, then there exists some $\epsilon > 0$ such that for $\vec{x_0} \in B_\epsilon(\vec{\alpha})$, the Newton's method iteration $\vec{x_k}$ converges quadratically to $\vec{\alpha}$.

Drawbacks of Newton's Method:

- ϵ is unknown.
- Relies on the invertability of J.
- Lack of robustness.
- Computation of derivatives is expensive.

Optimization

Generally, we can define an optimization problem in the following way:

$$\min_{(x,y,z,\ldots)} f(x, y, z, \ldots)$$

Subject to:

$$\vec{h_{eq}}(x, y, z, \ldots) = \vec{0} \qquad \qquad \vec{h}_{i-eq}(x, y, z, \ldots) \geq \vec{0}$$

To solve this problem, we could use Newton's method again, however, our linear algebra problem takes a different form:

$$H(x_k)\vec{\delta} = -\nabla f(x_k) \qquad \qquad x_{k+1} = x_k + t\delta$$

Our choice of t in this case can take many different forms:

- Exact Linesearch: $t = \underset{\tau}{\operatorname{argmin}}\, f(x_k + \tau\delta)$
- Fixed Linesearch
- Backtracking Linesearch

Quasi-Newton Methods: Often, computing H is expensive, or the backsolve comes with complications. Hence, we can define some other methods to subsitute our $H_k = H(x_k)$ for some B_k.

- $B_k = H(x_k)$ (Newton's Method)
- $B_k = H(x_k) + \lambda I$ (Regularized NM), most commonly $\lambda \geq -\sigma$.
- $B_k = I$ (Steepest Descenet)
- $B_{k+1} = B_k +$ "low rank update" (BFGS)

Interpolation

Notation: First, we define some notation:

$$\Pi_n = \{\text{All 1-D real polynomials of degree} \leq n\}$$

Lagrange Polynomial Interpolation Problem: Suppose we have $n+1$ distinct points $x_i, i \in 0, 1, \ldots, n$, and values $f_i = f(x_i)$ for some unknown function $f : \mathbb{R} \to \mathbb{R}$. We wish to find some function $g(x)$ that interpolates $f_i \forall i$.

Cardinal Polynomials: We can define the *Cardinal Polynomials* for a set of points as the following:

$$L_{n,k}(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)(x_k-x_1)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)}$$

For all $k \in 0, 1, \ldots, n$.
We note that:

- $L_{n,k}(x_i) = 0, \forall i \neq k$.
- $L_{n,k}(x_k) = 1$

Hence, we can define the interpolating polynomial in the following way:

$$p_n \equiv \sum_{k=0}^{n} f_i L_{n,k}(x) \in \Pi_n$$

It is known that for these interpolating polynomials that the following conditions hold:

- Uniqueness (By Contradiction)
- Existence

Error in Lagrange Polynomial Interpolation:

Theorem: Suppose f has $n+1$ continuous derivatives in (x_0, x_n) and p_n is the interpolating polynomial through $x_0 < x_1 < \ldots < x_n$ (ordered), then $\exists\, \xi = \xi(x)$ such that:

$$e(x) = f(x) - p_n(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_n)}{(n+1)!} f^{(n+1)}(\xi(x))$$

Proof. (sketch) We define some function $\phi(t)$:

$$\phi(t) \equiv e(t) - \frac{e(x)}{\pi(x)}\pi(t)$$

If we take that $\phi(t)$ has $n+2$ zeroes, and p_n has n zeroes, then it follows that taking successive derivatives of $\phi(t)$ yields the following:

$$\phi^{(n+1)}(t) = e^{(n+1)}(t) - \frac{e(x)}{\pi(x)}\pi^{(n+1)}(t)$$

However, since $e^{(n+1)}(t) = f^{(n+1)}(t) - p_n^{(n+1)}(t)$ and $p_n^{(n+1)}(t) = 0$ since $p_n \in \Pi_n$, then $e^{(n+1)}(t) = f^{(n+1)}(t)$. Hence, if we take some $\xi(x)$ where $\phi^{(n+1)}(t) = 0$, then we get that:

$$0 = \phi^{(n+1)}(\xi(x)) = f^{(n+1)}(\xi(x)) - \frac{e(x)}{\pi(x)}(n+1)!$$

$$\implies e(x) = \frac{\pi(x)}{(n+1)!} f^{(n+1)}(\xi(x))$$

□

Chebyshev Points: Equispaced points can often to lead to large errors in interpolation. Hence, we can employ *Chebyshev Points*, which follow from the x components of points on the unit circle.

An aside on instability and stability: Small errors can be magnified by an unstable algorithm, even for a well-conditioned problem!

Barycentric Formula for Lagrange Polynomial Interpolation: This can prove to be a much more stable method of computing Lagrange polynomial interpolants.

$$\begin{aligned} p_n(x) &= \frac{\sum_{k=0}^{n} \frac{w_k}{x-x_k} f_k}{\sum_{k=0}^{n} \frac{w_k}{x-x_k}} \\ w_k &= \frac{1}{(x_k - x_0)(x_k - x_1)\cdots(x_k - x_{k-1})(x_k - x_{k+1})\cdots(x_k - x_n)} \end{aligned}$$

There exists other methods as well, such a recursive approach.

Quadrature

Problem: Suppose we have data on equispaced points/nodes/grid. We wish th approximate the definite integral:

$$\int_{x_0=a}^{x_n=b} f(x)dx$$

We will explore two different ways of computing this integral:

1. Construction of an interpolant, then integrating the interpolant.
2. Subdividing the domain and computing a discrete composite sum.

Starting off with case (1), formally, we are saying that we would like to demonstrate the following:

Note: the following section is heavily taken from the professor's notes.

Given $f(x_k)$ at $n+1$ equally spaced points $x_k = x_0 + k \cdot h, k = 0, 1, \ldots, n$, where $h = \frac{(x_n - x_0)}{n}$, and suppose that $p_n(x)$ interpolates the data. Hence, we wish to find some form such that:

$$\int_{x_0}^{x_n} f(x)dx \approx \int x_0{}^{x_n} p_n(x)dx$$

We can note that:

$$\begin{aligned}\int_{x_0}^{x_n} p_n(x)dx &= \int_{x_0}^{x_n} \sum_{k=0}^{n} f(x_k) \cdot L_{n,k}(x)dx \\ &= \sum_{k=0}^{n} f(x_k) \cdot L_{n,k}(x)dx = \sum_{k=0}^{n} w_k f(x_k)\end{aligned}$$

where the coefficients

$$w_k = \int_{x_0}^{x_n} L_{n,k}(x)dx$$

$k = 0, 1, \ldots, n$ are independent of f.

A formula of the form:

$$\int_a^b f(x)dx \approx \sum_{k=0}^{n} w_k f(x_k)$$

with $x_k \in [a, b]$ is called a *quadrature formula*, with the coefficients w_k known as *weights*.
There are two common quadrature formulae, called the Newton-Cotes formulae:

- Trapezoidal Rule ($n = 1$):

  $$\int_{x_1}^{x_0} f(x)dx \approx \frac{h}{2}\left[f(x_0) + f(x_1)\right]$$

  Error:

  $$e_1(x) \leq \frac{(x_1 - x_0)^3}{12} \max_{\xi \in [x_0, x_1]} |f''(\xi)|$$

- Simpson's Rule ($n = 2$):

  $$\int_{x_0}^{x_2} f(x)dx \approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right]$$

Error:

$$\begin{aligned} e_2(x) &\leq \frac{(x_2-x_0)^5}{720} \max_{\xi\in[x_0,x_2]} |f''''(\xi)| \\ &\leq \frac{x_2-x_0}{6}\left[f(x_0)+4f(x_1)+f(x_2)\right] - \frac{(x_2-x_0)^5}{2880} f''''(\xi) \end{aligned}$$

Composite Quadrature: Knowing that the formulations in (1) only account for 1 or 2 point cases, we can guess that as n increases, we will be susceptible to the Runge phenomenon. Hence, instead of increasing the degree of the polynomial, we subdivide the domain of integration: split $[a, b]$ into n equal intervals $[x_{i-1}, x_i]$, for $i = 1, \ldots, n$.
For the integral between the subdivisions, we can use a Newton-Cotes formula:

- Trapezoidal Rule

$$\int_{x_0}^{x_n} f(x)dx = \sum_{i=1}^{n} \left[\frac{h}{2}\left[f(x_{i-1}+x_i)\right] - \frac{h^3}{12} f''(\xi_i)\right]$$

- Simpson's Rule

$$\int_{x_0}^{x_{2n}} f(x)dx = \sum_{i=1}^{n} \left[\frac{h}{3}\left[f(x_{2i-2})+4f(x_{2i-1})+f(x_{2i})\right] - \frac{(2h)^5}{2880} f''''(\xi_i)\right]$$

Errors in Composite Quadrature:

- Composite Trapezoidal Rule:

$$e_n^{\mathrm{TR}} = \frac{(b-a)}{12} h^2 f''(\xi)$$

- Composite Simspon's Rule:

$$e_h^{\mathrm{S}} = \frac{(b-a)h^4}{180} f''''(\xi)$$

Adaptivity: It can be observed that specifying n nor h are not very intuitive, nor user friendly. Instead, it would be easier to specify a *tolerance*. We can take steps towards deriving a tolerance in the following example:

$$\begin{aligned} S_n - S_{n/2} &= I + e_n^{\mathrm{S}} - (I + e_{n/2}^{\mathrm{S}}) \\ &\approx e_n^{\mathrm{S}} - \frac{1}{16} e_n^{\mathrm{S}} = \frac{15}{16} e_n^{\mathrm{S}} \end{aligned}$$

From an algorithmic standpoint, this would mean that if we want an absolute error of ϵ, we would want to compute the sequence $S_n, S_{\frac{1}{2}n}, S_{\frac{1}{4}n}, \ldots$ and then stop when the difference between the two consecutive values is smaller than $\frac{16}{15}\epsilon$. This will insure that approximately $|e_h^{\mathrm{S}}| \leq \epsilon$.

Finite Differences

Like we did with functions earlier, we can discretize the second derivative with something called a *differencing scheme*:

$$u_{xx}(x_j) = \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + \mathcal{O}(h^2)$$

Matrix Form: Following from the earlier discretizations, if we have some x on an interval $[a, b]$, with boundary values of $u(a) = u_0 = \alpha$ and $u(b) = u_{m+1} = \beta$, then we can construct a matrix representation:

$$u_{xx} \approx \frac{1}{h^2} \begin{bmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & \ddots & & & & \\ & & & \ddots & & \\ & & & & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix} + \begin{bmatrix} \frac{\alpha}{h^2} \\ 0 \\ \vdots \\ 0 \\ \frac{\beta}{h^2} \end{bmatrix}$$

If we wanted to solve the steady problem $u_{xx} = f$, we would have that:

$$\frac{1}{h^2} \begin{bmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & \ddots & & & & \\ & & & \ddots & & \\ & & & & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix} = \begin{bmatrix} f_1 - \frac{\alpha}{h^2} \\ f_2 \\ \vdots \\ f_{m-1} \\ f_m - \frac{\beta}{h^2} \end{bmatrix}$$

which would be an easily backsolve-able problem.
Local Truncation Error (LTE): For the steady state problem $u_{xx} = f$, we can get the LTE τ at point x_j:

$$\tau_j = \frac{1}{h^2} \left[u(x_{j-1}) - 2u(x_j) + u(x_{j+1})) \right] - f(x_j)$$

More generally, for some scheme $s(t_i, ...) = f(u(t_j))$ we can derive the LTE in the following manner:

1. Setting it up as the stepwise difference: $\tau_j = s(t_i, ...) - f(u(t_j))$
2. Taylor expand the differencing scheme and look for the original problem, i.e. $u = f$ to cancel out.
3. Cancel out terms until you get a big-O form.

Consistency: We say a method is *consistent* if the LTE $\tau_j \to 0$, as $h \to 0$. We can do this by using the Taylor Series. Generally, terms will cancel, and one must also look out for the original problem popping up in the expansion, i.e. $u''(x_j) - f(x_j) = 0$, since $u_{xx} = f$.

Convergence: *Convergence* is when error $E \to 0$, as $h \to 0$. $\tau \to 0$ does not imply convergence. We can see this in the following relationship:

$$E = D^{-1}\vec{\tau}$$

Hence, if we wish to have that $E \to 0$, we need certain properties of the matrix D, provided that our method is consistent.

Stability: We can say that a method is *stable* if it does not amplify small errors. For the case of a linear BVP with form $DU = F$, then we can say that there exists some constant C such that:

$$\left\|(D_h)^{-1}\right\| \leq C$$

for sufficiently small h.
Given that $\|D\| = \max(\lambda) = \lambda_{\max}$, it follows that $\left\|D^{-1}\right\| = \frac{1}{\lambda_{\min}}$. Hence, we wish that $\lambda_{\min} \neq 0$ as $h \to 0$.

Dalhquist Theorem: Every one-step method that is consistent $\left(\lim_{k\to\infty} \tau^i \to 0\right)$, is convergent.

Forms of Stability (For ODEs)

Zero Stability: Apply the method to the problem: $u' = 0$, $u(0) = 1$.

- Definition: if U^n does not grow, we call the method zero-stable. If U^n grows in n, it is not zero-stable.
- Theorem: All consistent one-step methods are zero-stable.
- Method: Let $U^{(n)} = \xi^n$ as a solution to the problem. Insert ξ^n as a solution into the differencing scheme, and solve for ξ. If $|\xi| \leq 1$, then the method is zero-stable.
  - Note that if $\xi = 1$, it is required that ξ_i are unique, that is, that there are no repeated roots.

Dahlquist Test Problem: Let $u' = \lambda u$, $u(0) = 1$, $\lambda \in \mathbb{C}$. We expect the solution to take the form $u(t) = \exp(\lambda t)$. The method is as follows:

1. Write out the problem with the differencing scheme.
2. Substitute $f(U^{(i)})$ for λU^i.
3. Let $z = k\lambda$ and substitute.
4. Factor out some U^j such that you have a non-variable exponent polynomial of U, and that the entire equation $= 0$..
5. Let $U = G$. Solve for the roots of G to find $G(z)$, or the *growth factor*.
6. Given this $G(z)$ function, you can conclude different forms of stability:
   - A-Stability or Absolute Stability: Need that $|G_1(z)| \leq 1$ and $|G_2| \leq 1$, and $G_1(z) \neq G_2(z)$.
     - We can then discuss the stability region as:

       $$\{z : |G_1(z)| \leq 1 \wedge |G_2(z)| \leq 1\}$$

     - Caution: if there are repeated roots, that is $G_1(z) = G_2(z)$, then this does not hold!
   - L-Stability or Asymptotic Stability: Requires that $G(z) \to 0$ as $\text{Re}(z) \to -\infty$.

Stiffness:

Methods for Solving PDEs

Full Discretization: Take for example, the heat equation, where we have that $u_t = u_{xx}$. We can proceed like we did in prior methods, by treating u_{xx} as an operator on u. In this case, let L be the finite differencing matrix for the second derivative in space. Hence, it folows that we can do the following:

- Forward Euler:

$$v^{n+1} = v^n + kLv^n$$

- Backward Euler:

$$v^{n+1} = v^n + kLv^{n+1}$$

Method of Lines: A similar concept, we can use finite differencing to make a system of ODEs:

$$u_t = u_{xx} \to \vec{v}(t)' = L\vec{v}(t).$$

Hence, one is left with a system of ODEs in each "row", which can then be solved for using whichever method you choose to step in time.

Stability in Method of Lines: One can combine the stability criteria from the time-differencing method, as well as the space-differencing (matrix) to get a relationship between your spatial spacing h and time spacing k.
von Neumann Stability Analysis: We can analyze the stability of a PDE method given the fully-discretized method, with the following procedure:

1. With the full-discretized method, make the following substitution:

$$v_j^n = \exp(i\xi x_j) = \exp(i\xi jh)$$

2. Factor out $\exp(i\xi h)$ to get an equation of the form:

$$v_j^{n+1} = g(\xi)v_j^n$$

   We then call the function $g(\xi)$ the *amplification factor*.

3. Analyzing the amplification factor $g(\xi)$, we attempt to get it into a sinusoidal form, such that we can place a bound on $g(\xi)$.

4. Finally, we can rearrange and do more analysis to find inequalities of h and k which satisfy $|g(\xi)| \leq 1$.

Floating Point Arithmetic

Reals are represented discretely on computers. Numbers are spaced evenly amongs intervals of powers of two, i.e; $\{[1,2),[2,4),[4,16),\ldots\}$. Hence, for each of these intervals, there are the same number of discrete representations.

Storage:

- 1 floating point number = 64 bits = 8 bytes
  - 1 bit for sign
  - 11 bits for exponent (base 2)
  - 53 bits for fraction (52 stored, 1 implicit)

We then call the distance between two numbers “machine epsilon”, which is: $2^{-52} \approx 2.2 \times 10^{-16}$.

IEEE Standard: According to the IEEE standard for floating point computing, each calculation should obey the following:

$$a?b = (a?b)(1+\epsilon)$$

Where ? is some operator, and $|\epsilon| \leq \epsilon_{\text{mach}}/2 \approx 1.1 \times 10^{-16}$. Hence, for every operation, it is quantized to the nearest discrete machine value.

Subtraction of Nearly Equal Numbers: When subtracting two nearly equal numbers, we lose out on bits of precision/information in the binary representation of the number. Hence, we wish to avoid this as much as possible. For example:

```
octave:30> x = 103.42
x = 0100000001011001110110101110000101000111101011100001010001111011
octave:31> y = x + 10 * eps(x)
y = 0100000001011001110110101110000101000111101011100001010010000101
octave:32> y - x
ans = 0011110101000100000000000000000000000000000000000000000000000000
octave:33> format long g
octave:34> y - x
ans = 1.4210854715202e-13
```

So instead of having order 10^{-16} precision in the operation, we only have 10^{-13}.

Backwards Stability Analysis: A backwards stable algorithm is one that gives the exact solution to a nearby problem, i.e.:

$$\texttt{sin}(x) = \sin(x+\delta)$$

Example: $\sin(n\pi)$ should be zero. However, $\sin\big(10^{20}\pi\big) = -0.394$. However, it holds that:

$$\sin\big(10^{20}\pi + \delta\big) = \sin\big(10^{20}\pi + 0.4\big) = 0$$

And hence, we can say that the *relative pertubation* is still small:

$$\frac{\delta}{|x|} = \frac{0.4}{10^{20}} = 4 \times 10^{-21}.$$

Spectral Methods

We can exploit the Fourier transform in order to solve differential equations. This allows one to have a method that has “infinite accuracy”, and through the convolution, uses all points in the domain. Using Fourier transforms generally has the following procedure:

1. Compute the FFT.
2. Multiply the FFT by $(ik)^n$ to take the n-th derivative.
3. Take the IFFT to get out the differentiated function.

Example: Solve $u_t = u_{xx} + u^2$. For every timestep, we do the following:

$$\begin{aligned} w &= \text{ifft}\left((ik)^2 \cdot \text{fft}(u)\right) \\ u_{n+1} &= u_n + \Delta t(w + u_n^2) \end{aligned}$$

Time Complexity: $\mathcal{O}(N \log N)$.

Limitations: When using spectral methods and the Fourier transform, one must be concerned with the following:

- *Periodicity of Boundary Conditions*: The Fourier transform needs periodic boundary conditions. Hence, one must extend their function to support this, and this extension should be well behaved/smooth (see next).
- *Smoothness of Function*: Due to Gibb’s Phenomena, discontinuous or “sharp” functions tend to not be well-behaved under the Fourier transform. Hence, using spectral methods is not recommended for these types of functions or solutions.

Radial Basis Functions

Instead of the the approaches seen with Lagrange interpolants earlier with polynomials, we can suggest other methods of interpolating functions. One of these methods is by means of *Radial Basis Functions*, which use radial distance to the points as a primary input:

$$S(\vec{x}) = \sum_{i=1}^{N} \lambda_i \phi\left(\|\vec{x} - \vec{x}_i\|_2\right).$$

Where:

- ϕ is some radial function (i.e. Gaussian)
- $\vec{x}_i$ are sample points
- λ_i are weights

And hence, this all becomes a linear algebra problem:

$$\begin{bmatrix} \phi(\|\vec{x}_1 - \vec{x}_1\|) & \phi(\|\vec{x}_1 - \vec{x}_2\|) & \phi(\|\vec{x}_1 - \vec{x}_3\|) & \cdots & \phi(\|\vec{x}_1 - \vec{x}_N\|) \\ \phi(\|\vec{x}_2 - \vec{x}_1\|) & \phi(\|\vec{x}_2 - \vec{x}_2\|) & \phi(\|\vec{x}_2 - \vec{x}_3\|) & \cdots & \phi(\|\vec{x}_2 - \vec{x}_N\|) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi(\|\vec{x}_N - \vec{x}_1\|) & \phi(\|\vec{x}_N - \vec{x}_2\|) & \phi(\|\vec{x}_N - \vec{x}_3\|) & \cdots & \phi(\|\vec{x}_N - \vec{x}_N\|) \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix}$$

We call the matrix with all of the ϕs in it the “colocation matrix” A.

Common Radial Functions:

- Infinity Smooth Radial Functions
  - GA: Gaussian $\phi(r) = e^{-\epsilon^2 r^2}$
  - IQ: Inverse Quadratic $\phi(r) = \frac{1}{1+e^2r^2}$
  - IMQ: Inverse Multiquadratic $\phi(r) = \frac{1}{\sqrt{1+\epsilon^2 r^2}}$
- Piecewise Smooth Radial Functions
  - Polyharmonic Splines: $\phi(r) = r^k$, k odd
  - Thin Plate Spline: $\phi(r) = r^2 \log r$

There are other methods, but the prof likes “RBF Finite Differences” the best.

Accuracy/Error: The error of RBFs is of order $\mathcal{O}(e^{-CN})$

- Pros:
  - Flexible Geometry
  - No Meshing
  - Spectral Accuracy
  - Interpretable
- Cons:
  - Dense Matricies
  - Stability Issues due to condition number of A and the shape parameter/weights.

RBFs for Discretizing Differentiation Operators: We can use RBFs to approximate solutions for PDEs. Take some differential operator L, which could be anything like: $\frac{\partial}{\partial x}, \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$, etc. Hence, for some differential equation of the form:

$$Lu = g$$

We can then apply L to the interpolant:

$$B = L \circ A = \begin{bmatrix} L(\phi(\|x_1 - x_1\|)) & L(\phi(\|x_2 - x_1\|)) & \cdots & L(\phi(\|x_N - x_1\|)) \\ & \ddots & & \vdots \\ & & & L(\phi(\|x_N - X_N\|)) \end{bmatrix}$$

And then we can get that:

$$B\lambda = g$$

Knowing that $A\lambda = u$, then it follows that $\lambda = A^{-1}u$, so:

$$BA^{-1}u = g$$

This is known as Kansa’s method.

Conservation Laws

We find conservation law style PDEs in the form:

$$q_t + [f(q)]_x = 0$$

Where:

- q is a vector of conserved quantities
- $f(q)$ is a flux function

With initial conditions $q(x,0) = q_0(x)$.

Traffic Flow Problems: The classic form of these problems are traffic flow problems. I don't think we need to know too much, but just in case, here's the shock equation again:

$$s = \frac{f(q_l) - f(q_r)}{q_l - q_r}$$

We can come up with methods to solve these types of problems. First, consider the *integral form of the conservation law*:

$$\frac{d}{dt}\int_{x_l}^{x_r} q(x,t)dx = f(q(x_l,t)) - f(q(x_r,t)).$$

For methods to work, we divide the domain by midpoints on cells, instead of boundaries. Hence, we can call these "finite volume methods". From this, we can also define cell-centered averages:

$$Q_j(t) = \frac{1}{h}\int_{x_{j-\frac{1}{2}}}^{x_{j-\frac{1}{2}}} q(x,t)dx.$$

Finite volume methods find how these Q_js evolve over time.

Lax Friedrichs Method: From this, we get the *Lax-Friedrichs Method* for solving conservation law problems:

$$Q_j^{n+1} = \frac{1}{2}(Q_{j-1}^n - Q_{j+1}^n) - \frac{k}{2h}a\left(Q_{j+1}^n - Q_{j-1}^n\right).$$

Conservative Form: We can suggest a "simple and natural way" to ensure that we do non converge for non-solutions (particularly helpful for nonlinear problems with discontinuities):

$$Q_j^{n+1} = Q_j^n - \frac{k}{h}\left(F_{j+\frac{1}{2}}^n - F_{j-\frac{1}{2}}^n\right) = Q_j^n - \frac{k}{h}\left[F(Q_j^n, Q_{j+1}^n) - F(Q_{j-1}^n, Q_j^n)\right]$$

In this case, we call F the *numerical flux function.*

Numerical Flux Functions: are some mythical beasts that can be derived in different ways and can take different forms depending on the nature of the problem. At a basis, for consistency, it is required that:

$$F_{j+\frac{1}{2}}^n(u,u) = f(u).$$

Example: The numerical flux function in the Lax-Friedrichs Method is:

$$F_{j+\frac{1}{2}}^n = \frac{f(Q_{j+1}^n) + f(Q_j^n)}{2} + \frac{h}{k}\left(\frac{Q_{j+1}^n - Q_j^n}{2}\right)$$

We can also do better on these numerical flux functions, i.e., instead of taking just Q_{j+1}^n or Q_j^n, we could take a linear interpolation. Lax-Wendroff Theorem: If a consistent and conservative method converges, then it converges to a weak solution.

- *Consistency*: $F(u,u) = u$
- *Stability*: Not investigated in this course.

Norms: Max-norm error (∞-norm) is problematic for discontinuous solutions. Hence, a better choice is the functional 1-norm:

$$|v|_1 = \int_{-\infty}^{\infty} |v(x)| dx.$$

Riemann Solvers: Numerical methods/subroutines specialized for solving discontinuous problems.

Weak/Variational Forms of PDEs

Take a PDE of some form:

$$\begin{cases} Lu = f \\ u(a) = u(b) = 0 \end{cases} \tag{1}$$

Where L is some differential operator of highest degree 2, a classic solution (sometimes called a *strong solution*) to the PDE would be $u \in C^2([a,b])$ such that it satisfies (1). However, if we multiply the problem by some arbitrary function $\phi \in C^1([a,b])$, $\phi(a) = \phi(b) = 0$ and integrate, we can get something of the following form:

$$\int_a^b Lu(x)\phi(x) = \int_a^b f(x)\phi(x)dx, \ \forall\, \phi \in C^1([a,b]),\, \phi(a) = \phi(b) = 0 \tag{2}$$

Since this is a general form, L is not defined, but, in general, integration by parts and the boundary conditions are employed to shift derivatives to other terms. Hence, if we can find some $u(x)$ that satisfies (2), then u is considered to be a *weak solution* of the PDE (1).

Notation: We define some notation for variational problems:

Inner Product	Bilinear Form
$(v,w) = \int_a^b v(x)w(x)dx$	$a(v,w) = \int_a^b v'(x)w'(x)dx$

And don't forget your integration by parts!!!

$$-\int_a^b u''(x)v(x) = -\left[u'(x)v(x)\right]_a^b + \int_a^b u'(x)v'(x)dx$$

Aside: Minimization Problem: Find $u \in V$ such that $F(u) \leq F(v)$ for all $v \in V$, where $F(v) = \frac{1}{2}a(v,v) - (f,v)$.

Finite Element Problems

Grid: Domain is divided into intervals that are not necessarily equal in size. We can define notation for everything:

- Domain divisions: $x_0 < x_1 < \ldots < x_M < x_{M+1}$
- Intervals: $I_j = (x_{j-1}, x_j)$
- Interval Length: $h_i = x_j - x_{j-1}$
- $h = \max h_j$

Basis Functions: Basis functions are the choice of v, however in this case (and from what I gather, most cases), the choice for $\phi_j(x)$ is the "hat function". Hence, for some value of h, we can write $u_h(x)$ as a weighted sum of the basis functions.

We denote the *stiffness matrix* A as:

$$a_{ij} = a(\phi_i, \phi_j) = (\phi_i', \phi_j'),$$

and the load vector b:

$$b_i = (f, \phi_i).$$

And then solve the following linear algebra problem:

$$A\xi = b$$

Where then the solution u can be reconstructed:

$$u_h(x) = \sum_{i=1}^{m} \xi_i \phi_i(x)$$

2D Case for the Poisson Problem: In the 2D case, we have the problem $-\nabla^2 u = f$ on a domain Ω and with $u = 0$ on the boundary $\partial\Omega = \Gamma$.

In this form, the weak form still is as follows:

$$a(u, v) = (f, v)$$

However, the bilinear form is different:

$$a(u, v) = \int_\Omega \nabla u \nabla v \, d\Omega$$

Now, instead of 1d line segment intervals, we have non-overlapping triangles K_i that are our "intervals". Additionally, instead of the "hat functions" that we had in the prior case, now we have "tent functions" to support the additional dimension.

- Tent functions have 1 at node j, and 0 at neighbours (and everywhere else). (hexagonal!)

Level Set Functions

Consider some function $\phi(x, y)$ which is continuous and has negative and positive values. Then, we can say that $\phi < 0$ is “inside” and $\phi > 0$ is “outside” and $\phi = 0$ is the “boundary”. With this, we can describe interface motion by use of PDEs:

- Advection by a velocity field: $\phi_t + V \cdot \nabla\phi = 0$
- Motion in the normal direction: $\phi_t + V_n\|\nabla\phi\|_2 = 0$.
- Motion by mean curvature: $\phi_t + \nabla \cdot \left(\frac{\nabla\phi}{\|\nabla\phi\|_2}\right)\|\nabla\phi\|_2 = 0$

Hamilton-Jacobi Equations: Hamilton-Jacobi equations bear the following form:

$$\phi_t + H(\nabla\phi) = 0.$$

Some level set equations are examples of HJ equations. In this case, H is a Hamiltonian.

There exists a connection between HJ equations and conservation laws. If one differentiates w.r.t. x and set $u = \phi_x$, then we get that:

$$u_t + H(u)_x = 0.$$

Thus, the solution to an HJ equation is the integral of a conservation law solution. HJ solutions are generally “kinked”.

Numerical Methods for HJ equations: Build a “numerical Hamiltonian”.

Numerical Linear Algebra

Notation:

$$M(i, j, \lambda) = I + \begin{pmatrix} 0 & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Where i is the row position and j is the column position.
Gaussian Elimination and Backsolving: For some 3x3 matrix, it follow that:

$$M(2, 1, -l_{21}) \cdot M(3, 1, -l_{31}) \cdot M(3, 2, -l_{32}) \cdot A = U \text{ (upper triangular)}$$

Hence, consequentially, we get that:

$$M(2, 1, -l_{21}) \cdot M(3, 1, -l_{31}) \cdot M(3, 2, -l_{32}) = L = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix}$$

Note: each value of l_{ij} is derived from each *successive* matrix multiplication.

$$l_{ij} = \frac{a_{ij}}{a_{jj}}$$

However, this can fail with zero division. Thus, we can do *partial pivoting.*