ApproxFun and Solution of Differential Equations in Julia

Sheehan Olver
The University of Sydney

- `ApproxFun` is based on computational Chebyshev and Fourier series
  - Similar to `Chebfun`
- Posing differential equations as acting on coefficients results in almost banded operators
  - This allows for solving equations in infinite dimensions
  - Doing this efficiently in a general framework depends on `Julia`'s multiple dispatch

Demo 1:
Basic ApproxFun features

Computational Fourier and Chebyshev Series

- The kth Fourier coefficient is

$$\hat{f}_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(\theta) \mathrm{e}^{-\mathrm{i}k\theta} \, \mathrm{d}x$$

- For f periodic and sufficiently smooth, we have the Fourier expansion

$$f(\theta) = \sum_{k=-\infty}^{\infty} \hat{f}_k \mathrm{e}^{\mathrm{i}k\theta}$$

- Our starting point is to use the FFT to approximate the coefficients $\hat{f}_k$

- Define the m evenly spaced points on the periodic interval $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$:

$$\boldsymbol{\theta} := \left(-\pi, \left(\frac{2}{m} - 1\right)\pi, \ldots, \left(1 - \frac{2}{m}\right)\pi\right) =$$

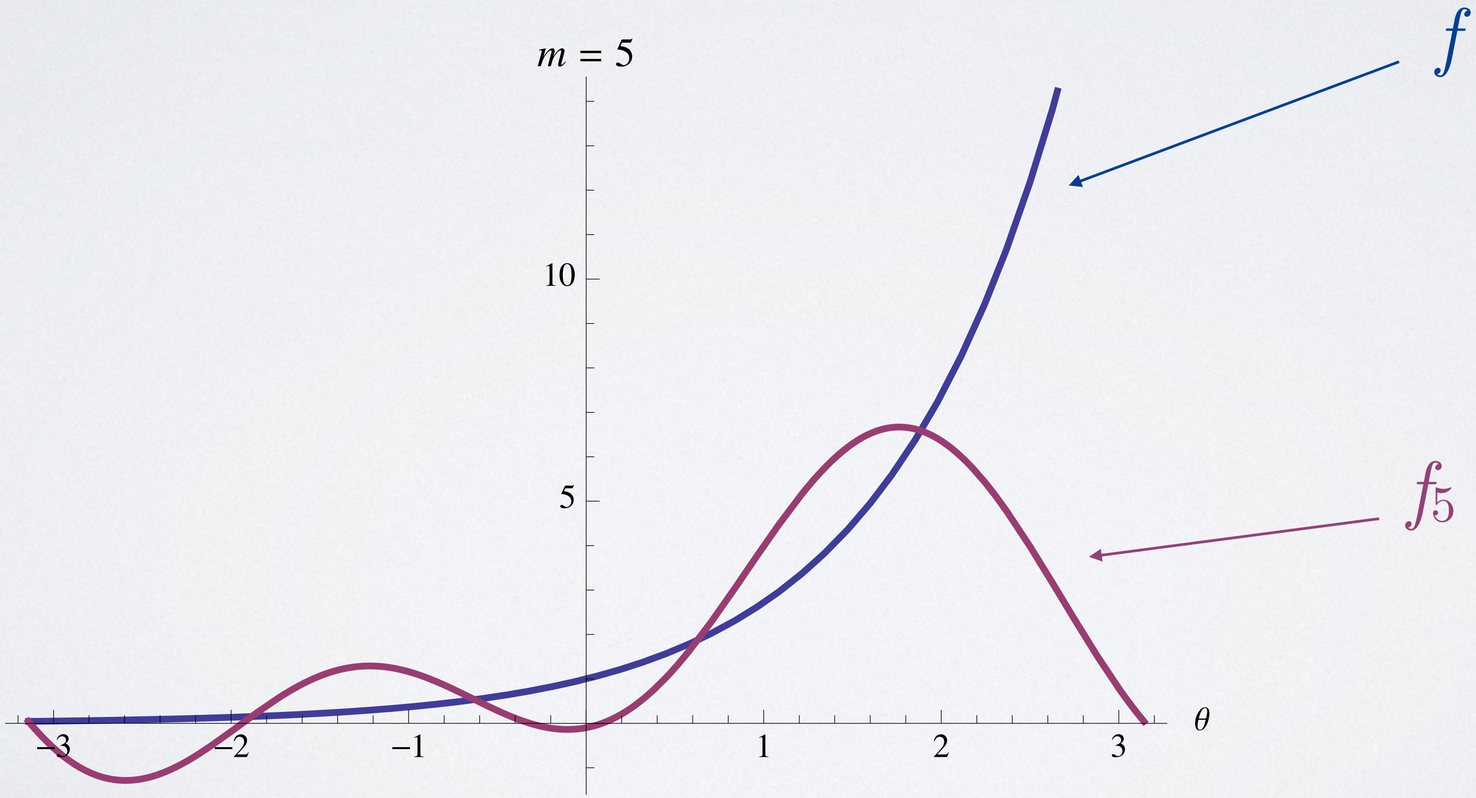
- We can approximate the Fourier coefficients using the m point trapezoidal rule

$$\hat{f}_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(\theta)\,\mathrm{d}\theta \approx \frac{1}{m}\sum_{j=1}^{m} f(\theta_j)\mathrm{e}^{-k\theta_j} =: \hat{f}_k^m$$

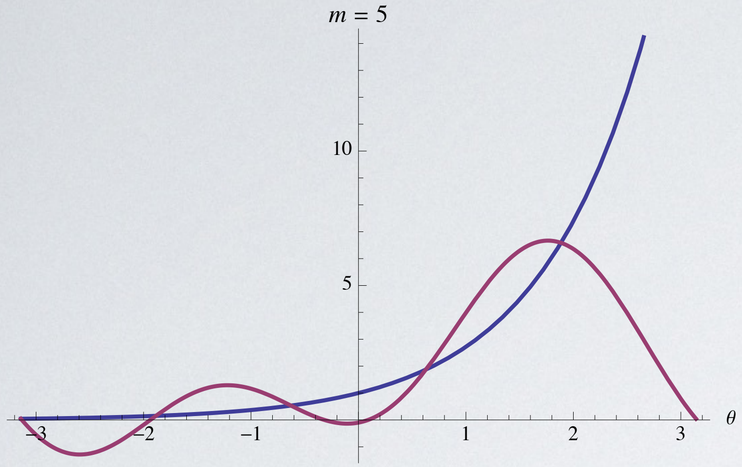
- Using the approximate Fourier coefficients, we get an approximation to f:

$$f_m(\theta) := \begin{cases} \sum_{k=-\frac{m-1}{2}}^{\frac{m-1}{2}} \hat{f}^m_k \mathrm{e}^{\mathrm{i}k\theta} & m \text{ odd} \\ \frac{\hat{f}^m_{-m/2}}{2}\mathrm{e}^{-\mathrm{i}\frac{m}{2}\theta} + \sum_{k=1-\frac{m}{2}}^{\frac{m}{2}-1} \hat{f}^m_k \mathrm{e}^{\mathrm{i}k\theta} + \frac{\hat{f}^m_{m/2}}{2}\mathrm{e}^{\mathrm{i}\frac{m}{2}\theta} & m \text{ even} \end{cases}$$

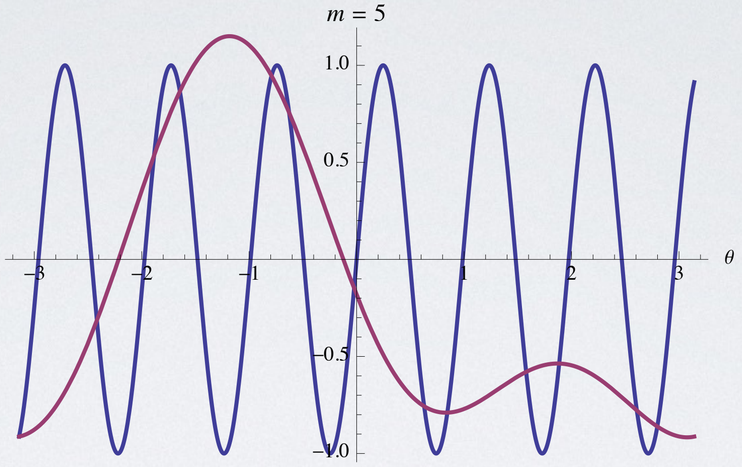
- This interpolates f at θ_j



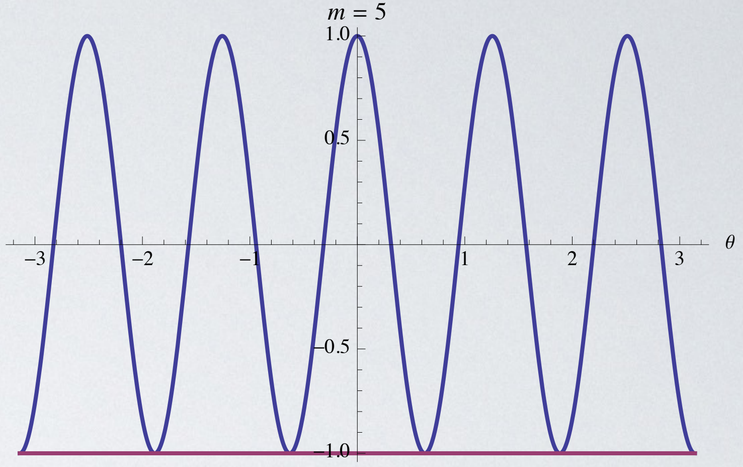
e^{θ}



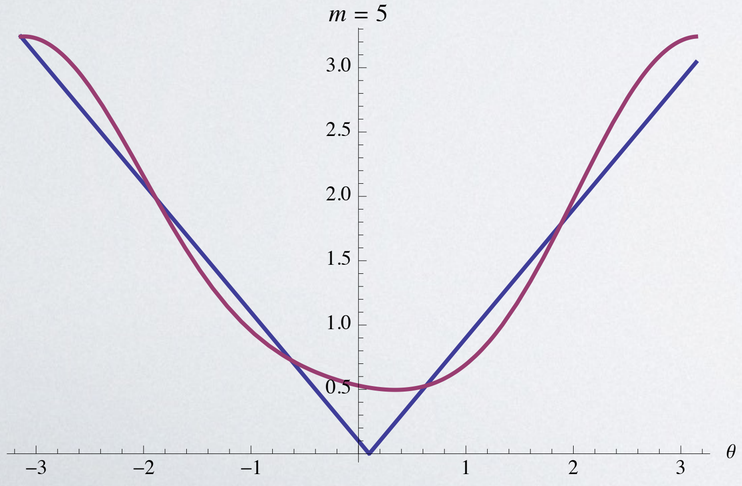
$\sin \frac{20}{\pi}\theta$



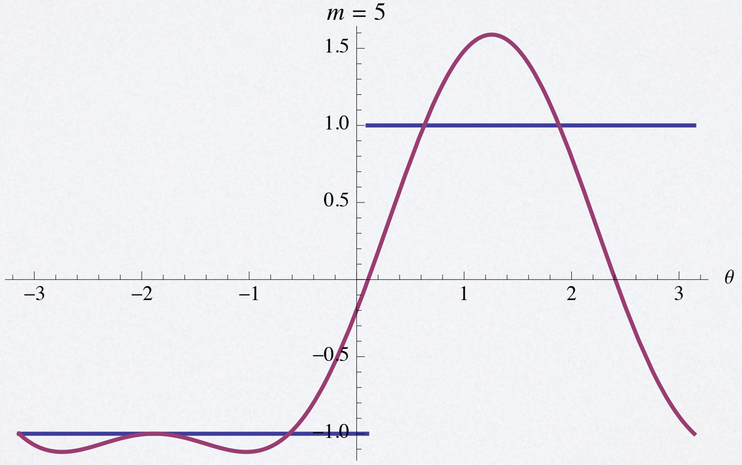
$\cos 5\theta$



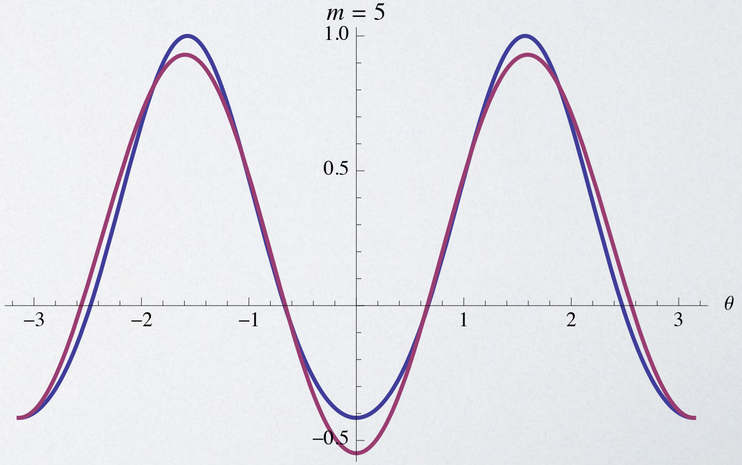
$|\theta - .1|$



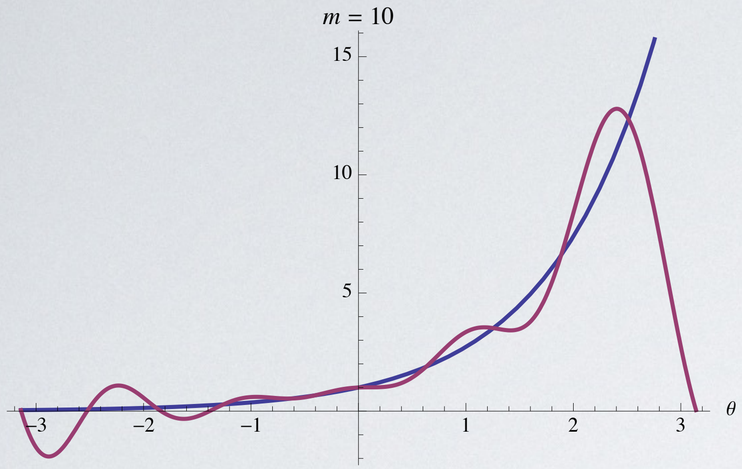
$\operatorname{sgn}(\theta - .1)$



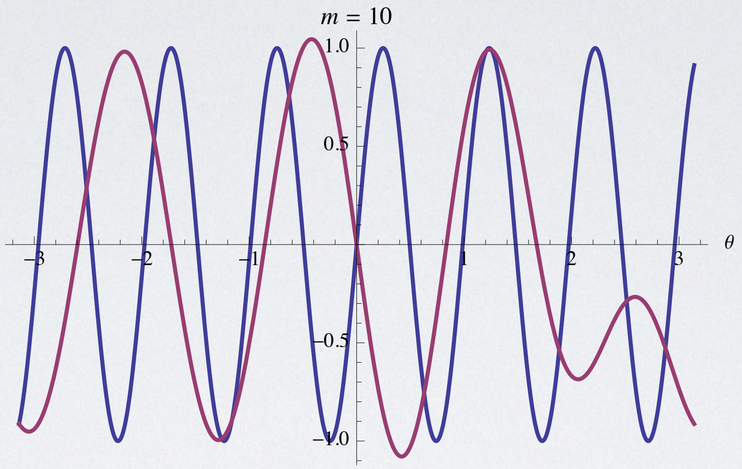
$\cos 2 \cos \theta$



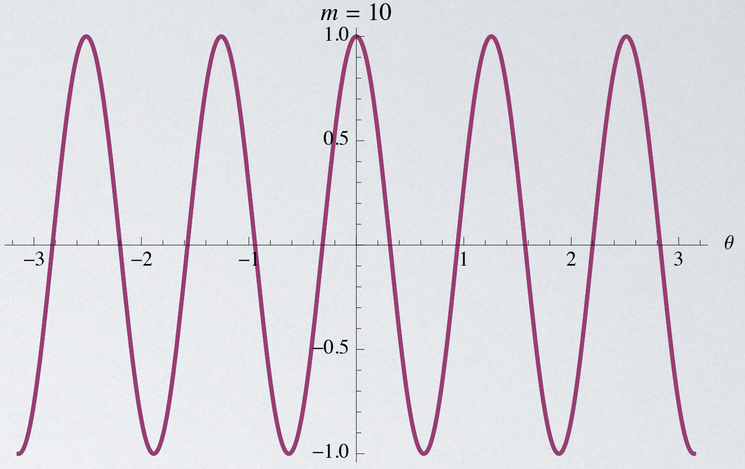
e^{θ}



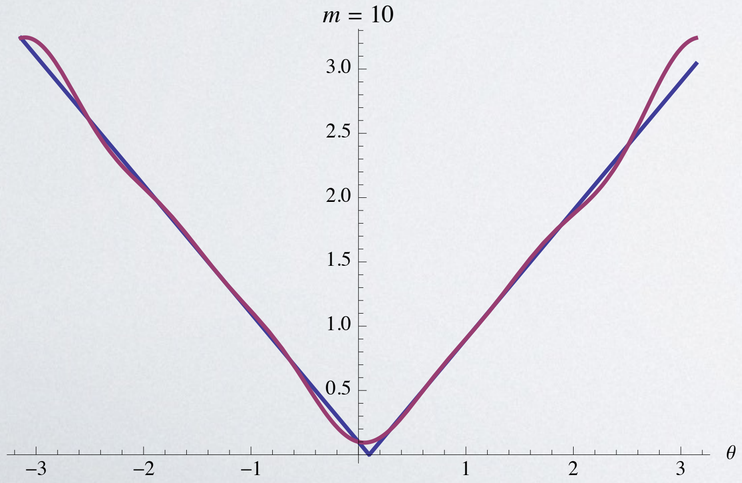
$\sin \frac{20}{\pi}\theta$



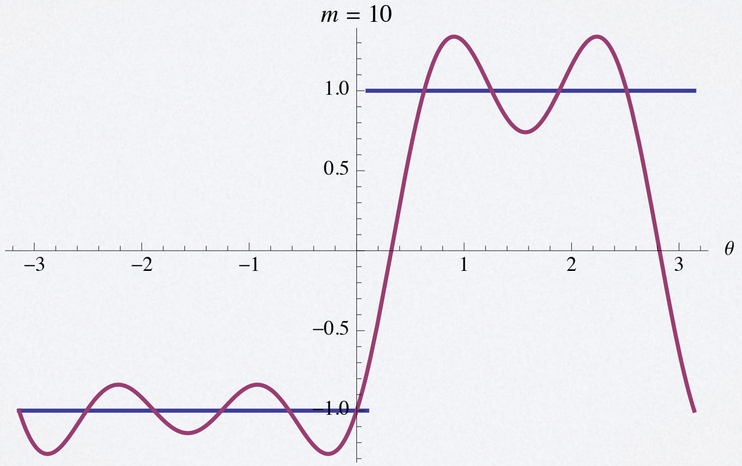
$\cos 5\theta$



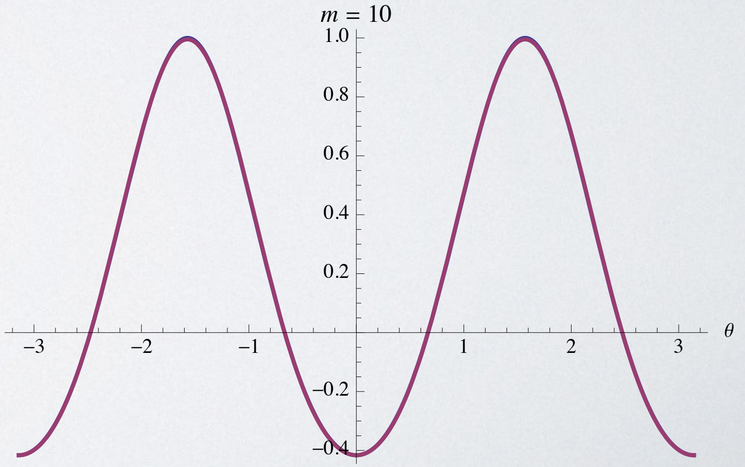
$|\theta - .1|$



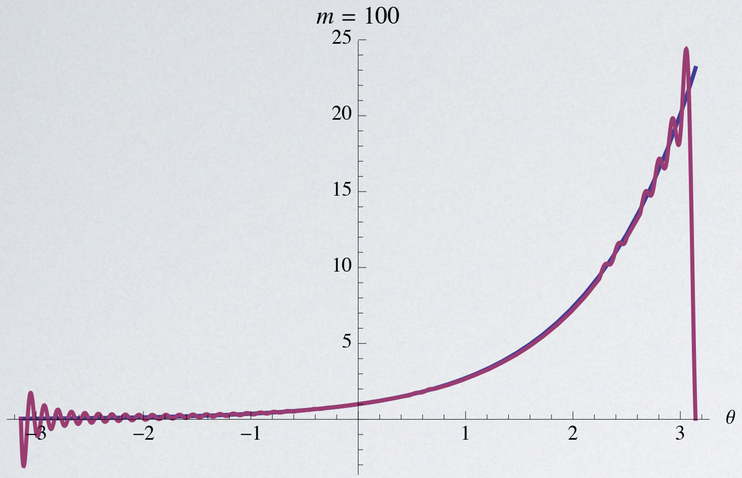
$\operatorname{sgn}(\theta - .1)$



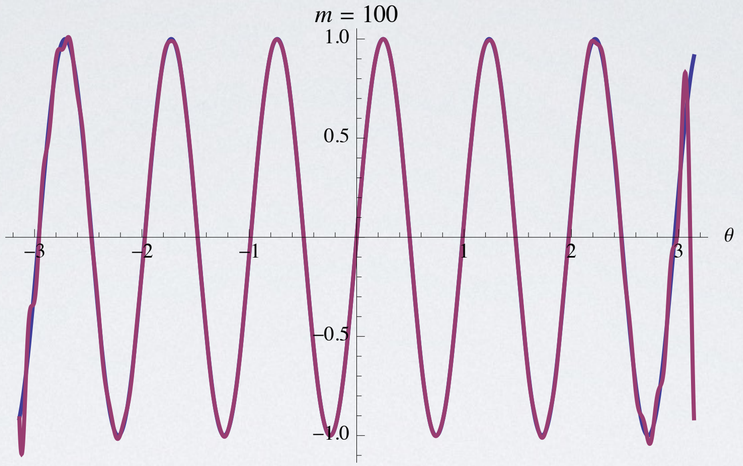
$\cos 2 \cos \theta$



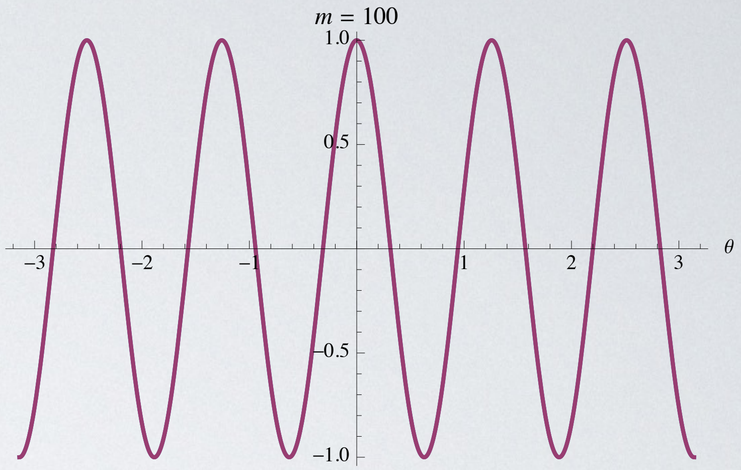
e^{θ}



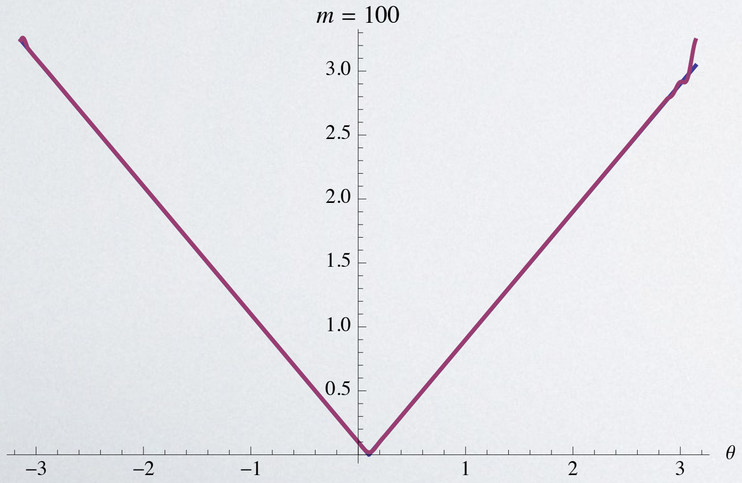
$\sin \frac{20}{\pi} \theta$



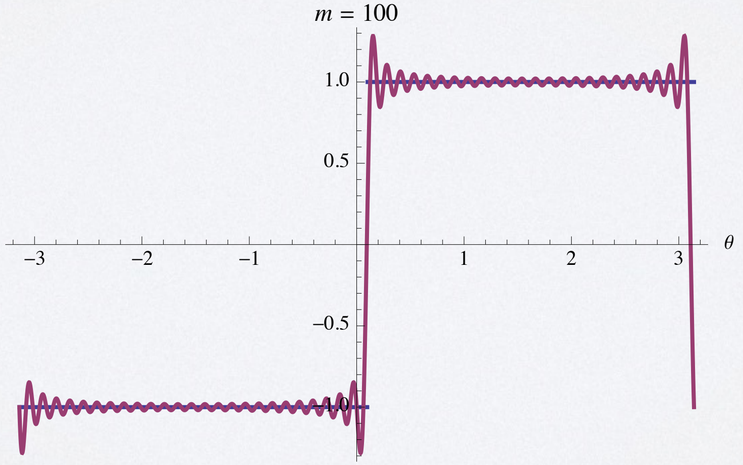
$\cos 5\theta$



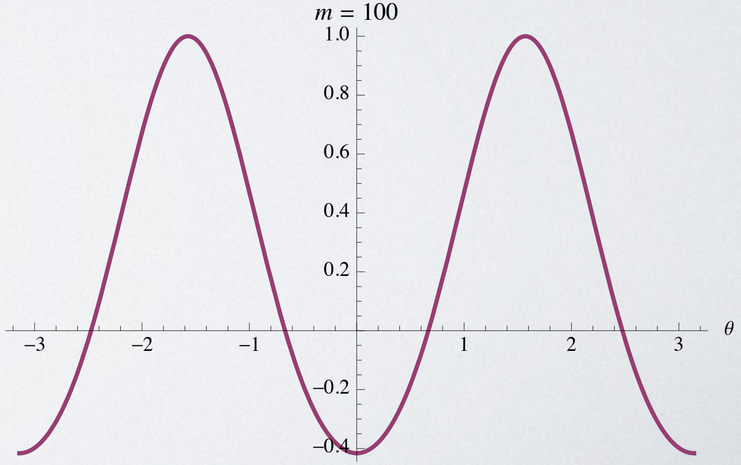
$|\theta - .1|$



$\operatorname{sgn}(\theta - .1)$

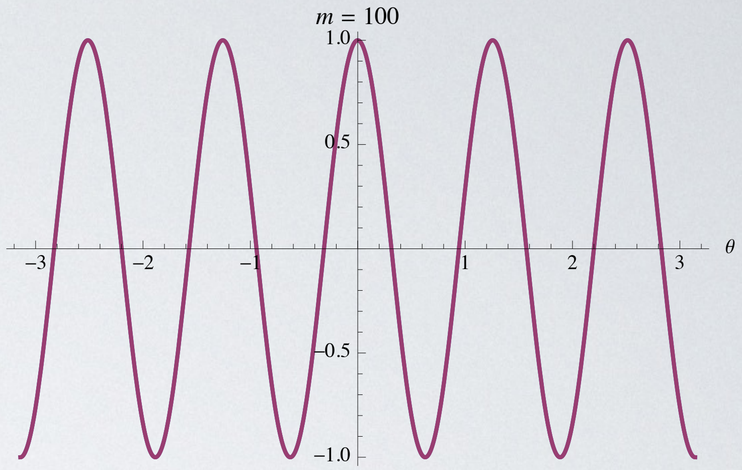


$\cos 2 \cos \theta$

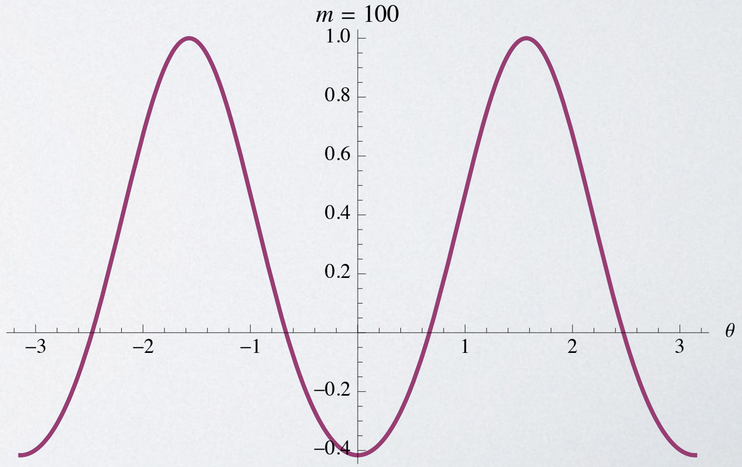


Smooth and periodic converge very fast!

$\cos 5\theta$

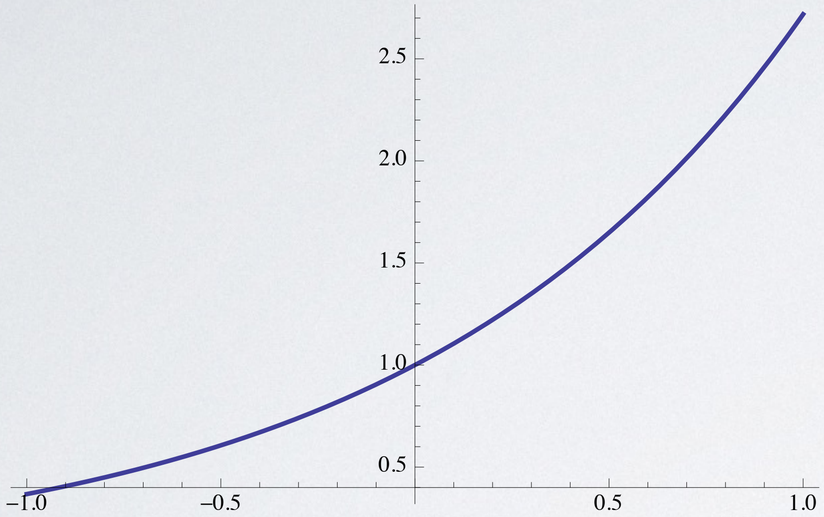


$\cos 2\cos\theta$



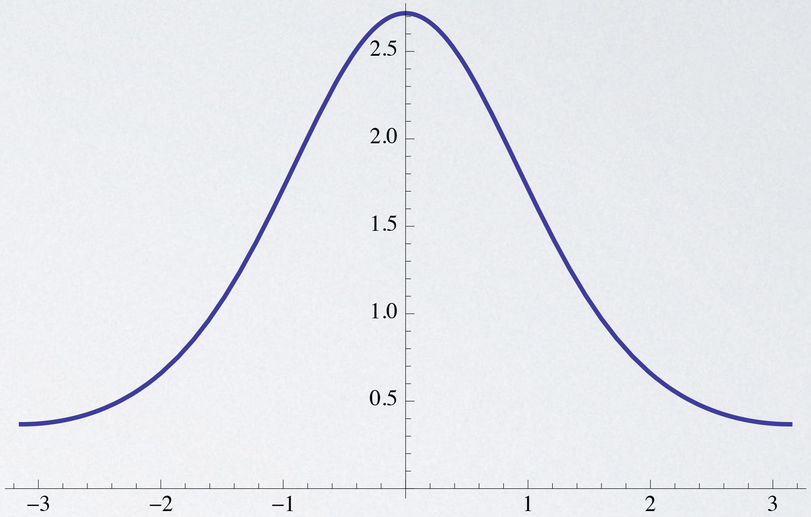
Non-periodic functions

$$e^{x}$$

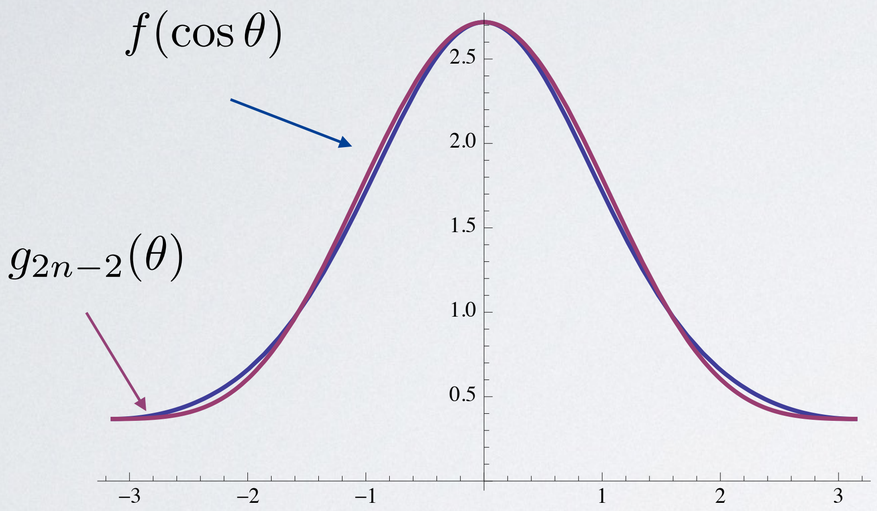


$$x = \cos\theta$$

$$e^{\cos\theta}$$



Smooth and periodic!



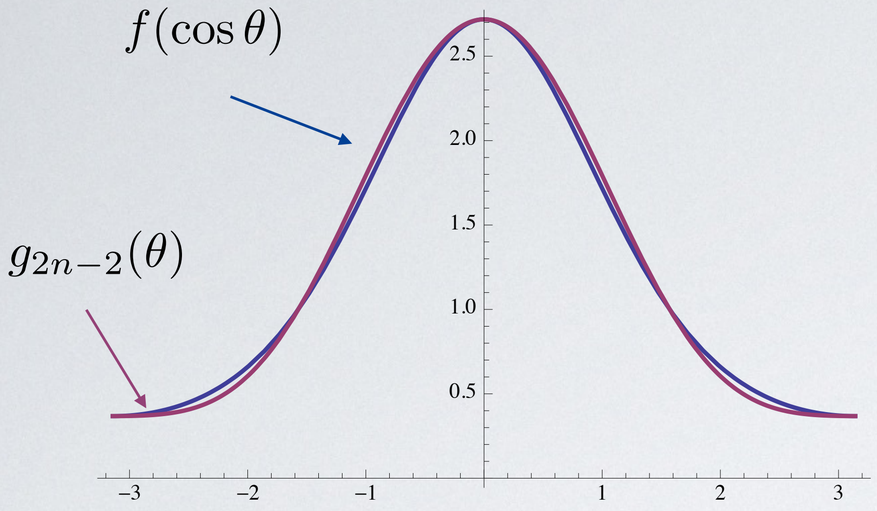
- Let $g(\theta) = f(\cos\theta)$ and approximate

$$g(\theta) \approx g_{2n-2}(\theta)$$

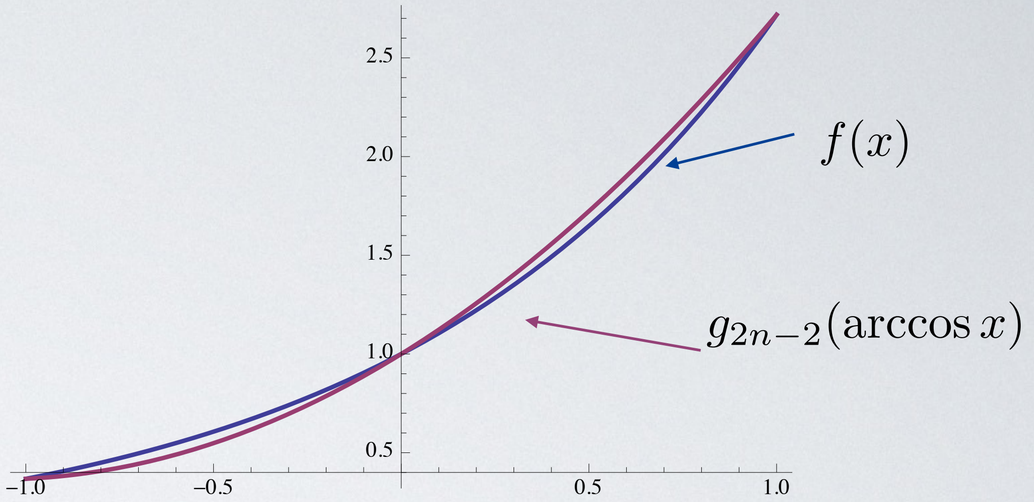
- Symmetry around zero implies that $\hat{g}_k^{2n-2} = \hat{g}_{-k}^{2n-2}$ giving us

$$g_{2n-2}(\theta) = \sum_{k=0}^{n-1} \check{f}_k^n \cos k\theta$$

(where $\check{f}_k^n$ are $\hat{g}_k^n$ but sometimes divided by 2)



$$\theta = \arccos x$$



$$f(x) \approx g_{2n-2}(\arccos x) = \sum_{k=0}^{n-1} \check{f}_k^n T_k(x) \quad \text{for} \quad T_k(x) := \cos k \arccos x$$

- We can replace functions by finite vector of coefficients!

- Thus we represent smooth functions $f(x)$ by a finite-dimensional vector of Chebyshev coefficients

$$f(x) \to \begin{pmatrix} \check{f}_0^n \\ \vdots \\ \check{f}_{n-1}^n \end{pmatrix}$$

  - The length n depends on f, and is chosen automatically by testing tail of coefficients for decay

- Standard function operations then translate to operations on variable length vectors

  - Evaluation is through *Clenshaw's algorithm*

  - Addition is pad to same length and add vectors

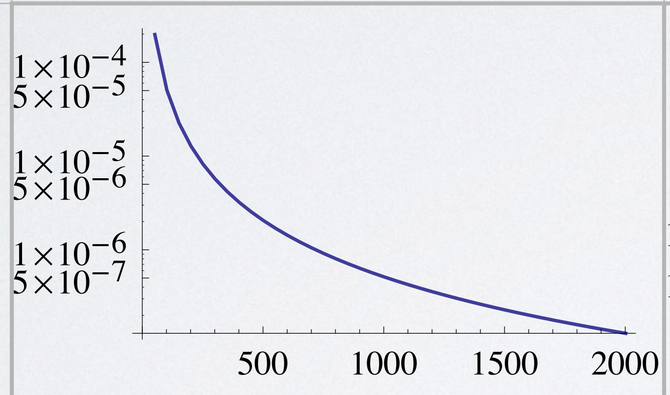
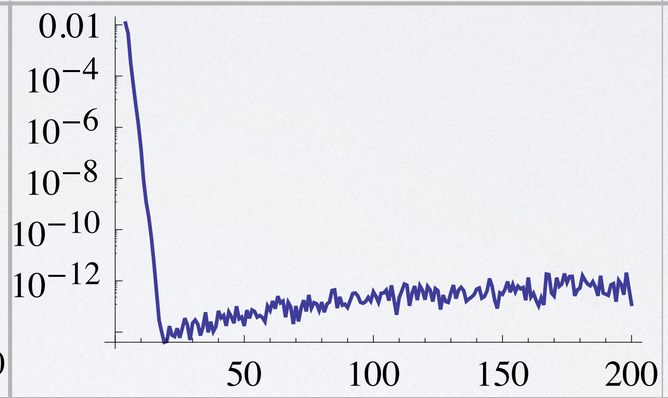
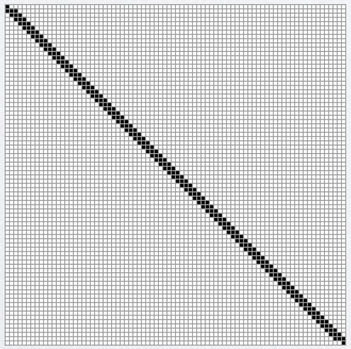
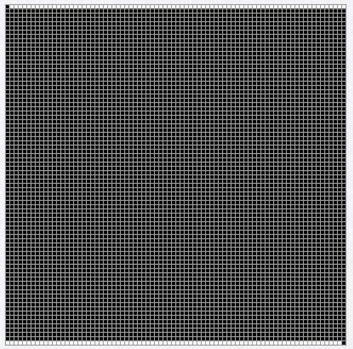
  - Multiplication is like a convolution of vectors

- Other operations are accomplished by inverting infinite-dimensional linear equations

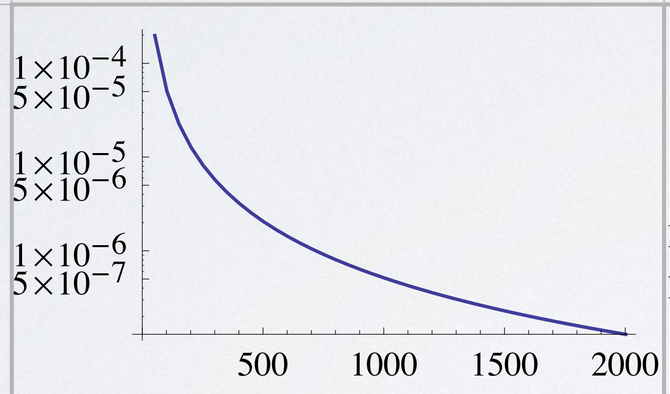
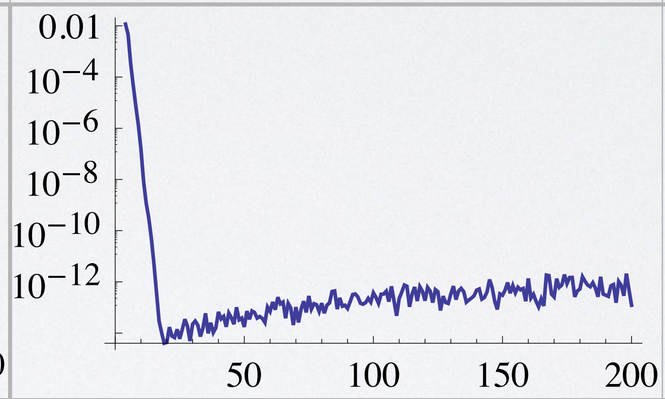
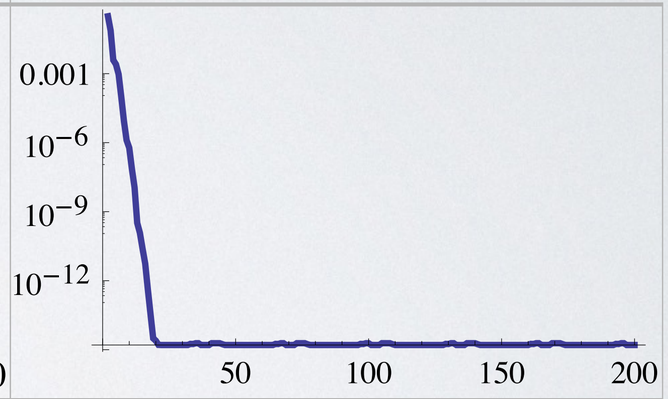
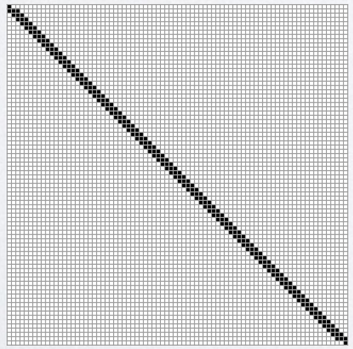
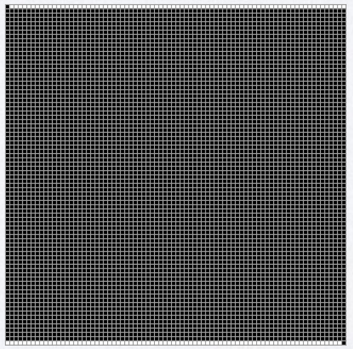
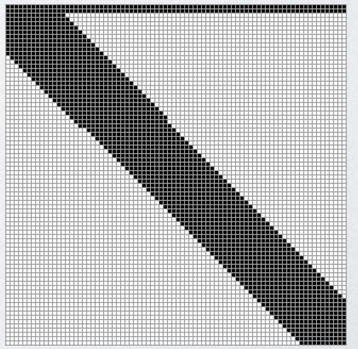
  - Solving differential equations, applying special functions and even division

Demo 2:
Differential equations

- Usual approach to ODEs (finite differences, finite elements, collocation methods): represent differential operator by action at finite number of points
- *Ultraspherical spectral methods* [Olver & Townsend 2013]: represent by action on vector of coefficients, and change basis to get banded operators, and solve in infinite dimensions

	Finite difference	Collocation (*Chebfun v4*)	Ultraspherical (*ApproxFun*)
Convergence			
Sparsity			

- Usual approach to ODEs (finite differences, finite elements, collocation methods): represent differential operator by action at finite number of points
- *Ultraspherical spectral methods* [Olver & Townsend 2013]: represent by action on vector of coefficients, and change basis to get banded operators, and solve in infinite dimensions

	Finite difference	Collocation (*Chebfun v4*)	Ultraspherical (*ApproxFun*)
Convergence			
Sparsity			

1ST ORDER EXAMPLE

$$u(1) = 1$$

$$u' + u = f$$

coefficients

Evaluation at 1

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & \cdots \\ \hline 1 & 1 & -\frac{1}{2} & & & & \\ & \frac{1}{2} & 2 & -\frac{1}{2} & & & \\ & & \frac{1}{2} & 3 & -\frac{1}{2} & & \\ & & & \frac{1}{2} & 4 & -\frac{1}{2} & \\ & & & & \ddots & \ddots & \ddots \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ \vdots \end{pmatrix} = \begin{pmatrix} 1 \\ \hline f_0 \\ \vdots \\ f_{n-1} \\ 0 \\ \vdots \end{pmatrix}$$

Instead of truncating and solving, we will solve in infinite-dimensions

$$u' + u = 2 + x \qquad \text{and} \qquad u(1) = 1$$

$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

Introduce zero using *Givens rotation* on first two rows

$$\begin{pmatrix} 1. & 1. & 1. & 1. & 1. & 1. & \dots \\ 1. & 1. & -0.5 & 0. & 0. & 0. & \dots \\ 0 & 0.5 & 2. & -0.5 & 0. & 0. & \dots \\ 0 & 0 & 0.5 & 3. & -0.5 & 0. & \dots \\ 0 & 0 & 0 & 0.5 & 4. & -0.5 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \mathtt{u} = \begin{pmatrix} 1 \\ 2 \\ 1 \\ 0 \\ 0 \\ \vdots \end{pmatrix}$$

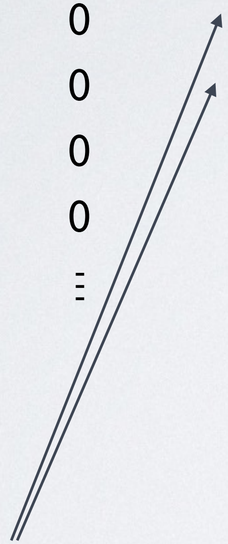
$$u' + u = 2 + x \qquad \text{and} \qquad u(1) = 1$$

$$\begin{pmatrix} 1.41421 & 1.41421 & 0.353553 & 0.707107 & 0.707107 & 0.707107 & \dots \\ 0 & 0 & -1.06066 & -0.707107 & -0.707107 & -0.707107 & \dots \\ 0 & 0.5 & 2. & -0.5 & 0. & 0. & \dots \\ 0 & 0 & 0.5 & 3. & -0.5 & 0. & \dots \\ 0 & 0 & 0 & 0.5 & 4. & -0.5 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \mathtt{u} = \begin{pmatrix} 2.12132 \\ 0.707107 \\ 1 \\ 0 \\ 0 \\ \vdots \end{pmatrix}$$

Now do *Givens rotation*
on next two rows

$$u' + u = 2 + x \qquad \text{and} \qquad u(1) = 1$$

$$\begin{pmatrix} 1.41421 & 1.41421 & 0.353553 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & \dots \\ 0 & 0.5 & 2. & -0.5 & 0. & 0. & 0. & \dots \\ 0 & 0 & 1.06066 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & \dots \\ 0 & 0 & 0.5 & 3. & -0.5 & 0. & 0. & \dots \\ 0 & 0 & 0 & 0.5 & 4. & -0.5 & 0. & \dots \\ 0 & 0 & 0 & 0 & 0.5 & 5. & -0.5 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \mathtt{u} = \begin{pmatrix} 2.12132 \\ 1. \\ -0.707107 \\ 0 \\ 0 \\ 0 \\ \vdots \end{pmatrix}$$



And so on

$$u' + u = 2 + x \qquad \text{and} \qquad u(1) = 1$$

$$\begin{pmatrix} 1.41421 & 1.41421 & 0.353553 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & \dots \\ 0 & 0.5 & 2. & -0.5 & 0. & 0. & 0. & 0. & \dots \\ 0 & 0 & 1.1726 & 1.91881 & 0.426401 & 0.639602 & 0.639602 & 0.639602 & \dots \\ 0 & 0 & 0 & 2.41209 & -0.753778 & -0.301511 & -0.301511 & -0.301511 & \dots \\ 0 & 0 & 0 & 0.5 & 4. & -0.5 & 0. & 0. & \dots \\ 0 & 0 & 0 & 0 & 0.5 & 5. & -0.5 & 0. & \dots \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 6. & -0.5 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \quad \mathtt{u} = \begin{pmatrix} 2.12132 \\ 1. \\ -0.639602 \\ 0.301511 \\ 0 \\ 0 \\ 0 \\ \vdots \end{pmatrix}$$

$$u' + u = 2 + x \qquad \text{and} \qquad u(1) = 1$$

$$\begin{pmatrix} 1.41421 & 1.41421 & 0.353553 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & \dots \\ 0 & 0.5 & 2. & -0.5 & 0. & 0. & 0. & 0. & 0. & \dots \\ 0 & 0 & 1.1726 & 1.91881 & 0.426401 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & \dots \\ 0 & 0 & 0 & 2.46337 & 0.0738088 & -0.396722 & -0.295235 & -0.295235 & -0.295235 & \dots \\ 0 & 0 & 0 & 0 & 4.06973 & -0.428393 & 0.061199 & 0.061199 & 0.061199 & \dots \\ 0 & 0 & 0 & 0 & 0.5 & 5. & -0.5 & 0. & 0. & \dots \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 6. & -0.5 & 0. & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 7. & -0.5 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \quad \mathtt{u} = \begin{pmatrix} 2.12132 \\ 1. \\ -0.639602 \\ 0.295235 \\ -0.061199 \\ 0 \\ 0 \\ 0 \\ \vdots \end{pmatrix}$$

$$u' + u = 2 + x \qquad \text{and} \qquad u(1) = 1$$

$$\begin{pmatrix}
1.41421 & 1.41421 & 0.353553 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & \dots \\
0 & 0.5 & 2. & -0.5 & 0. & 0. & 0. & 0. & 0. & 0. & \dots \\
0 & 0 & 1.1726 & 1.91881 & 0.426401 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & \dots \\
0 & 0 & 0 & 2.46337 & 0.0738088 & -0.396722 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & \dots \\
0 & 0 & 0 & 0 & 4.10033 & 0.18451 & -0.000228355 & 0.0607423 & 0.0607423 & 0.0607423 & \dots \\
0 & 0 & 0 & 0 & 0 & 5.01493 & -0.503731 & -0.00746269 & -0.00746269 & -0.00746269 & \dots \\
0 & 0 & 0 & 0 & 0 & 0.5 & 6. & -0.5 & 0. & 0. & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 7. & -0.5 & 0. & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 8. & -0.5 & \dots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{pmatrix}
\mathtt{u} =
\begin{pmatrix}
2.12132 \\
1. \\
-0.639602 \\
0.295235 \\
-0.0607423 \\
0.00746269 \\
0 \\
0 \\
0 \\
\vdots
\end{pmatrix}$$

$$u' + u = 2 + x \qquad \text{and} \qquad u(1) = 1$$

$$\begin{pmatrix}
1.41421 & 1.41421 & 0.353553 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & \dots \\
0 & 0.5 & 2. & -0.5 & 0. & 0. & 0. & 0. & 0. & 0. & 0. & \dots \\
0 & 0 & 1.1726 & 1.91881 & 0.426401 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & \dots \\
0 & 0 & 0 & 2.46337 & 0.0738088 & -0.396722 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & \dots \\
0 & 0 & 0 & 0 & 4.10033 & 0.18451 & -0.000228355 & 0.0607423 & 0.0607423 & 0.0607423 & 0.0607423 & \dots \\
0 & 0 & 0 & 0 & 0 & 5.03979 & 0.0940168 & -0.0570311 & -0.00742587 & -0.00742587 & -0.00742587 & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 6.02037 & -0.496793 & 0.000740377 & 0.000740377 & 0.000740377 & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 7. & -0.5 & 0. & 0. & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 8. & -0.5 & 0. & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 9. & -0.5 & \dots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{pmatrix}
\quad u = \quad
\begin{pmatrix}
2.12132 \\
1. \\
-0.639602 \\
0.295235 \\
-0.0607423 \\
0.00742587 \\
-0.000740377 \\
0 \\
0 \\
0 \\
\vdots
\end{pmatrix}$$

$$u' + u = 2 + x \qquad \text{and} \qquad u(1) = 1$$

$$\begin{pmatrix}
1.41421 & 1.41421 & 0.353553 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & \dots \\
0 & 0.5 & 2. & -0.5 & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & \dots \\
0 & 0 & 1.1726 & 1.91881 & 0.426401 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & \dots \\
0 & 0 & 0 & 2.46337 & 0.0738088 & -0.396722 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & \dots \\
0 & 0 & 0 & 0 & 4.10033 & 0.18451 & -0.000228355 & 0.0607423 & 0.0607423 & 0.0607423 & 0.0607423 & 0.0607423 & \dots \\
0 & 0 & 0 & 0 & 0 & 5.03979 & 0.0940168 & -0.0570311 & -0.00742587 & -0.00742587 & -0.00742587 & -0.00742587 & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 6.0411 & 0.0842762 & -0.0406453 & 0.000737837 & 0.000737837 & 0.000737837 & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 7.0171 & -0.498346 & -0.0000612783 & -0.0000612783 & -0.0000612783 & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 8. & -0.5 & 0. & 0. & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 9. & -0.5 & 0. & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 10. & -0.5 & \dots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{pmatrix}$$

$$u = \begin{pmatrix}
2.12132 \\
1. \\
-0.639602 \\
0.295235 \\
-0.0607423 \\
0.00742587 \\
-0.000737837 \\
0.0000612783 \\
0 \\
0 \\
0 \\
\vdots
\end{pmatrix}$$

- Suppose we wanted to do back substitution (but don't do it yet!)

$$\begin{pmatrix}
1.41421 & 1.41421 & 0.353553 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & \dots \\
0 & 0.5 & 2. & -0.5 & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & \dots \\
0 & 0 & 1.1726 & 1.91881 & 0.426401 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & \dots \\
0 & 0 & 0 & 2.46337 & 0.0738088 & -0.396722 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & \dots \\
0 & 0 & 0 & 0 & 4.10033 & 0.18451 & -0.000228355 & 0.0607423 & 0.0607423 & 0.0607423 & 0.0607423 & 0.0607423 & \dots \\
0 & 0 & 0 & 0 & 0 & 5.03979 & 0.0940168 & -0.0570311 & -0.00742587 & -0.00742587 & -0.00742587 & -0.00742587 & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 6.0411 & 0.0842762 & -0.0406453 & 0.000737837 & 0.000737837 & 0.000737837 & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 7.0171 & -0.498346 & -0.0000612783 & -0.0000612783 & -0.0000612783 & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 8. & -0.5 & 0. & 0. & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 9. & -0.5 & 0. & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 10. & -0.5 & \dots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{pmatrix}
u =
\begin{pmatrix}
2.12132 \\
1. \\
-0.639602 \\
0.295235 \\
-0.0607423 \\
0.00742587 \\
-0.000737837 \\
0.0000612783 \\
0 \\
0 \\
0 \\
\vdots
\end{pmatrix}$$

- What would the forward error be?
  - Precisely the left over term:

$$\begin{pmatrix}
0. \\
0. \\
0. \\
0. \\
0. \\
0. \\
0. \\
0.0000612783 \\
0 \\
0 \\
0 \\
\vdots
\end{pmatrix}$$

- Thus we know the forward error already, before we have even done the back substitution!

Implementation in Julia

- Abstract data type representing infinite-dimensional operators
  - Each operator knows how to write sub-slices to finite-dimensional matrices
- Algebraic operations of operators
  - We have a "`PlusOperator`" and "`TimesOperator`" that contain a list of operators that they add or multiply
  - Thus we build up a tree of operators
- Automatic conversion between different coefficient spaces
  - Won't go in to
- Implementation of Chebyshev specific differentiation, multiplication and integration operators

Example Operator: Derivative of Taylor series on circle of radius R

$$\begin{pmatrix} 0 & R^{-1} & & & & \\ & & 2R^{-2} & & & \\ & & & 3R^{-3} & & \\ & & & & 4R^{-4} & \\ & & & & & \ddots \end{pmatrix}$$

```
type TaylorDerivativeOperator <:BandedOperator{Float64}
     radius
end

ApproxFun.bandrange(::TaylorDerivativeOperator)=0:1   # diagonal and 1 superdiagonal

function ApproxFun.addentries!(T::TaylorDerivativeOperator,A::ShiftArray,kr::Range1)
    R=T.radius

    for k=max(kr[1],1):kr[end]
        A[k,1]=k/R^k            # super-diagonal
    end

    A
end
```

∞-DIMENSIONAL LINEAR ALGEBRA DATA-TYPE

$$\begin{pmatrix} 1.41421 & 1.41421 & 0.353553 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & \cdots \\ 0 & 0.5 & 2. & -0.5 & 0 & 0. & 0. & 0. & 0. & 0. & 0. & \cdots \\ 0 & 0 & 1.1726 & 1.91881 & 0.426401 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & \cdots \\ 0 & 0 & 0 & 2.46337 & 0.0738088 & -0.396722 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & \cdots \\ 0 & 0 & 0 & 0 & 4.10033 & 0.18451 & -0.000228355 & 0.0607423 & 0.0607423 & 0.0607423 & 0.0607423 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 5.03979 & 0.0940168 & -0.0570311 & -0.00742587 & -0.00742587 & -0.00742587 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 6.02037 & -0.496793 & 0.00074037 & 0.000740377 & 0.000740377 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 7. & -0.5 & 0. & 0. & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 8. & -0.5 & 0. & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 9. & -0.5 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \end{pmatrix} \quad u = \begin{pmatrix} 2.12132 \\ 1. \\ -0.639602 \\ 0.295235 \\ -0.0607423 \\ 0.00742587 \\ -0.000740377 \\ 0 \\ 0 \\ 0 \\ \vdots \end{pmatrix}$$

Banded array

∞-DIMENSIONAL LINEAR ALGEBRA DATA-TYPE

$$\begin{pmatrix} 1.41421 & 1.41421 & 0.353553 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & \cdots \\ 0 & 0.5 & 2. & -0.5 & 0. & 0. & 0. & 0. & 0. & 0. & 0. & \cdots \\ 0 & 0 & 1.1726 & 1.91881 & 0.426401 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & \cdots \\ 0 & 0 & 0 & 2.46337 & 0.0738088 & -0.395523 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & \cdots \\ 0 & 0 & 0 & 0 & 4.10033 & 0.18451 & -0.0002 & 0.0607423 & 0.0607423 & 0.0607423 & 0.0607423 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 5.03979 & 0.0940168 & -0.0 & -0.00742587 & -0.00742587 & -0.00742587 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 6.02037 & -0.496793 & 0.000740377 & 0.000740377 & 0.000740377 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 7. & -0.5 & 0. & 0. & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 8. & -0.5 & 0. & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 9. & -0.5 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \end{pmatrix} u = \begin{pmatrix} 2.12132 \\ 1. \\ -0.639602 \\ 0.295235 \\ -0.0607423 \\ 0.00742587 \\ -0.000740377 \\ 0 \\ 0 \\ 0 \\ \vdots \end{pmatrix}$$

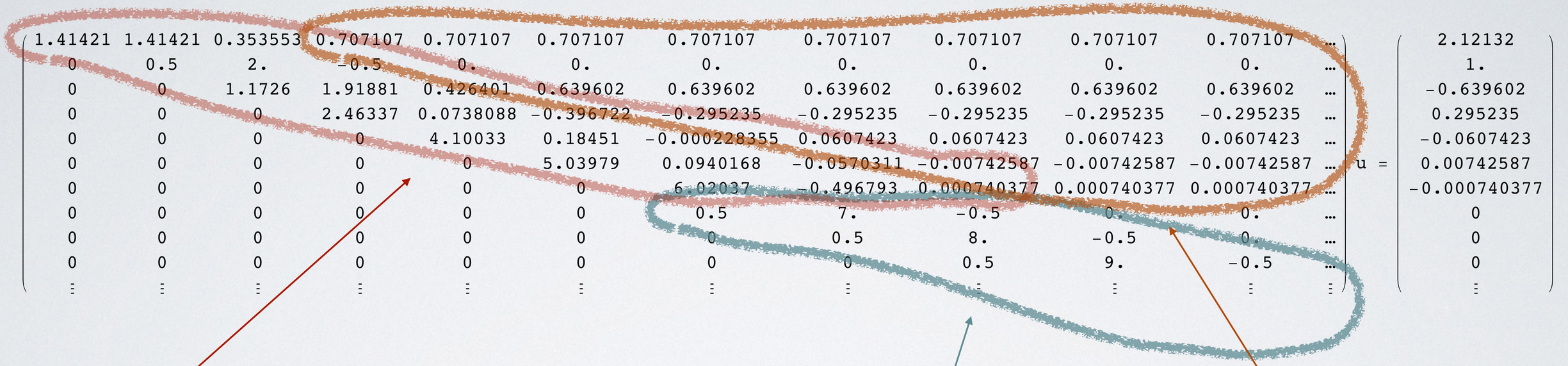
Fill-in by constant multiple of boundary row

∞-DIMENSIONAL LINEAR ALGEBRA DATA-TYPE

$$\begin{pmatrix}
1.41421 & 1.41421 & 0.353553 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & 0.707107 & \cdots \\
0 & 0.5 & 2. & -0.5 & 0. & 0. & 0. & 0. & 0. & 0. & 0. & \cdots \\
0 & 0 & 1.1726 & 1.91881 & 0.426401 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & 0.639602 & \cdots \\
0 & 0 & 0 & 2.46337 & 0.0738088 & -0.396722 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & -0.295235 & \cdots \\
0 & 0 & 0 & 0 & 4.10033 & 0.18451 & -0.000228355 & 0.0607423 & 0.0607423 & 0.0607423 & 0.0607423 & \cdots \\
0 & 0 & 0 & 0 & 0 & 5.03979 & 0.0940168 & -0.0570311 & -0.00742587 & -0.00742587 & -0.00742587 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 6.02827 & -0.496793 & 0.000740377 & 0.000740377 & 0.000740377 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 7. & -0.5 & 0. & 0. & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 8. & -0.5 & 0. & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 9. & -0.5 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{pmatrix}
u =
\begin{pmatrix}
2.12132 \\
1. \\
-0.639602 \\
0.295235 \\
-0.0607423 \\
0.00742587 \\
-0.000740377 \\
0 \\
0 \\
0 \\
\vdots
\end{pmatrix}$$

Unmodified banded operator

∞-DIMENSIONAL LINEAR ALGEBRA DATA-TYPE



Banded array

Unmodified banded operator

Fill-in by constant multiple of boundary row

Why Julia?

- Originally implemented in C++, but not easy to construct operators, or use the implementation
- Matlab is pass by value: no fast operators writing to matrices
  - Plus I hate Matlab..., and its OOP is ridiculously slow
- Python is too slow for linear algebra, and the nature of the data structure doesn't allow for passing off to C

Demo 3:
Partial Differential Equations

- We can solve general linear PDEs on a rectangle achieving machine precision accuracy
  - The current approach achieves $\mathcal{O}(n^3)$ operations for n^2 unknowns
  - Is there a fast, spectrally accurate Poisson solver on a rectangle to get $\mathcal{O}(n^2)$ operations??
- We do not know yet how to do PDEs in an "infinite-dimensional" way
- We also want to be able to scale to many sub-domains
  - Ideally, we would get $\mathcal{O}(mn^3)$ operations for m sub-domains
  - `Julia` will prove critical for large scale problems

Julia wish-list

- Fast 3D plotting
- iPad support
- A special matrix-type for banded matrices
- A low-level mode
  - Tired of debugging failed type inference
  - The old NumericExtensions `unsafe_view` was twice as fast as `@inbounds`
- A high-level mode that automatically devectorizes
  - I think `@devec` only works for simple assignment loops
- Typed and compiled anonymous functions