# 8. Chebyshev Series and the FFT

In this chapter we will see how Chebyshev spectral methods can be implemented by the FFT, which provides a crucial speedup for some calculations. Equally important will be the mathematical idea that underlies this technique: the equivalence of

| | |
|---|---|
| Chebyshev series | in $x \in [-1,1]$, |
| Fourier series | in $\theta \in \mathbb{R}$, |
| Laurent series | in $z$ on the unit circle. |

The basis of our development is summarized in Figure 8.1. Let $z$ be a complex number on the unit circle: $|z| = 1$. Let $\theta$ be the argument of $z$, a real number that is determined up to multiples of $2\pi$. Let $x = \operatorname{Re} z = \cos\theta$. For each $x \in [-1,1]$, there are two complex conjugate values of $z$, and we have

$$x = \operatorname{Re} z = \tfrac{1}{2}(z + z^{-1}) = \cos\theta \in [-1,1]. \tag{8.1}$$

The $n$th *Chebyshev polynomial*, denoted $T_n$, is defined by

$$T_n(x) = \operatorname{Re} z^n = \tfrac{1}{2}(z^n + z^{-n}) = \cos n\theta. \tag{8.2}$$

From this formula, it is not obvious that $T_n(x)$ is a polynomial in $x$. The cases $n = 0, 1, 2$, and 3 make the point clear:

$$\begin{aligned}
&\operatorname{Re} z^0 = 1 \quad \Rightarrow && T_0(x) = 1,\\
&\operatorname{Re} z^1 = \tfrac{1}{2}(z + z^{-1}) \quad \Rightarrow && T_1(x) = x,\\
&\operatorname{Re} z^2 = \tfrac{1}{2}(z^2 + z^{-2}) = \tfrac{1}{2}(z^1 + z^{-1})^2 - 1 \quad \Rightarrow && T_2(x) = 2x^2 - 1,\\
&\operatorname{Re} z^3 = \tfrac{1}{2}(z^3 + z^{-3}) = \tfrac{1}{2}(z^1 + z^{-1})^3 - \tfrac{3}{2}(z + z^{-1}) \quad \Rightarrow && T_3(x) = 4x^3 - 3x.
\end{aligned}$$

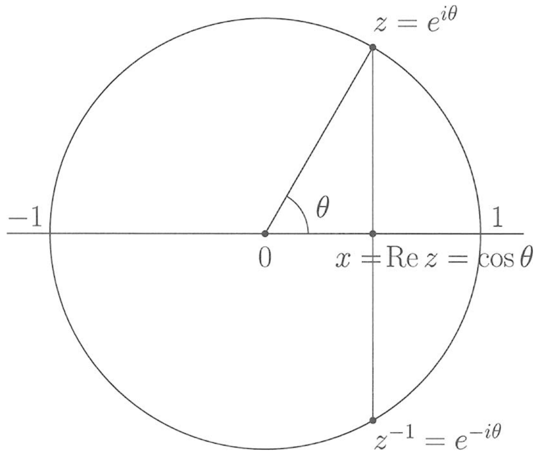


Fig. 8.1. *Relationships between $x$, $z$, and $\theta$.*

In general,

$$T_{n+1}(x) = \tfrac{1}{2}(z^{n+1} + z^{-n-1}) = \tfrac{1}{2}(z^n + z^{-n})(z + z^{-1}) - \tfrac{1}{2}(z^{n-1} + z^{1-n}),$$

which amounts to the recurrence relation

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x). \tag{8.3}$$

By induction, we deduce that $T_n$ is a polynomial of degree exactly $n$ for each $n \geq 0$, with leading coefficient $2^{n-1}$ for each $n \geq 1$. Figure 8.2 gives a geometric interpretation.

Since $T_n$ is of exact degree $n$ for each $n$, any degree $N$ polynomial can be written uniquely as a linear combination of Chebyshev polynomials,

$$p(x) = \sum_{n=0}^{N} a_n T_n(x), \qquad x \in [-1, 1]. \tag{8.4}$$

Corresponding to this is a degree $N$ *Laurent polynomial* in $z$ and $z^{-1}$ that is *self-reciprocal*, which means that $z^n$ and $z^{-n}$ have equal coefficients:

$$\mathsf{p}(z) = \tfrac{1}{2}\sum_{n=0}^{N} a_n(z^n + z^{-n}), \qquad |z| = 1. \tag{8.5}$$

Also corresponding to these is a degree $N$ $2\pi$-periodic trigonometric polynomial that is even, that is, such that $P(\theta) = P(-\theta)$:

$$P(\theta) = \sum_{n=0}^{N} a_n \cos n\theta, \qquad \theta \in \mathbb{R}. \tag{8.6}$$

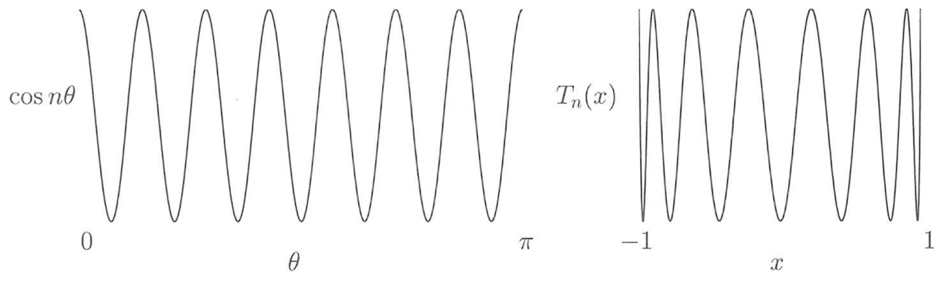


Fig. 8.2. *The Chebyshev polynomial $T_n$ can be interpreted as a sine wave "wrapped around a cylinder and viewed from the side."*

The functions (8.4)–(8.6) are equivalent in the sense that $p(x) = \mathsf{p}(z) = P(\theta)$ when $x$, $z$, and $\theta$ are related by (8.1). Note that we have introduced different fonts to distinguish the $x$, $z$, and $\theta$ domains. Similarly, from an arbitrary function $f(x)$ defined for $x \in [-1, 1]$, we can form a self-reciprocal function $\mathsf{f}(z)$ defined on the unit circle and a periodic function $F(\theta)$ defined on $\mathbb{R}$:

$$\mathsf{f}(z) = f\left(\frac{z + z^{-1}}{2}\right), \qquad F(\theta) = f(\cos\theta).$$

For spectral collocation methods, we mainly deal with (8.4)–(8.6) as interpolants of function $f$, $\mathsf{f}$, and $F$. The interpolation points are as follows:

$$\begin{aligned} \theta_j &= j\pi/N, \\ z_j &= e^{i\theta_j}, \\ x_j &= \cos\theta_j = \operatorname{Re} z_j, \end{aligned}$$

with $0 \le j \le N$. We have the equivalences:

$$P(\theta) \text{ interpolates } F(\theta) \text{ (even and } 2\pi\text{-periodic) in the equispaced points } \{\theta_j\}$$
$$\Updownarrow$$
$$\mathsf{p}(z) \text{ interpolates } \mathsf{f}(z) \text{ (self-reciprocal) in the roots of unity } \{z_j\}$$
$$\Updownarrow$$
$$p(x) \text{ interpolates } f(x) \text{ (arbitrary) in the Chebyshev points } \{x_j\}.$$

We are now prepared to describe an FFT algorithm for Chebyshev spectral differentiation. The key point is that the polynomial interpolant $q$ of $f$ can be differentiated by finding a trigonometric polynomial interpolant $Q$ of $F$, differentiating in Fourier space, and transforming back to the $x$ variable. Once we are working on a periodic equispaced grid, we can take advantage of the FFT.

---

## Chebyshev spectral differentiation via FFT

- Given data $v_0, \ldots, v_N$ at Chebyshev points $x_0 = 1, \ldots, x_N = -1$, extend this data to a vector $V$ of length $2N$ with $V_{2N-j} = v_j$, $j = 1, 2, \ldots, N-1$.
- Using the FFT, calculate
$$\hat{V}_k = \frac{\pi}{N} \sum_{j=1}^{2N} e^{-ik\theta_j} V_j, \qquad k = -N+1, \ldots, N.$$
- Define $\hat{W}_k = ik\hat{V}_k$, except $\hat{W}_N = 0$.
- Compute the derivative of the trigonometric interpolant $Q$ on the equispaced grid by the inverse FFT:
$$W_j = \frac{1}{2\pi} \sum_{k=-N+1}^{N} e^{ik\theta_j} \hat{W}_k, \qquad j = 1, \ldots, 2N.$$
- Calculate the derivative of the algebraic polynomial interpolant $q$ on the interior grid points by
$$w_j = -\frac{W_j}{\sqrt{1-x_j^2}}, \qquad j = 1, \ldots, N-1,$$
with the special formulas at the endpoints
$$w_0 = \frac{1}{2\pi} {\sum_{n=0}^{N}}' n^2 \hat{v}_n, \qquad w_N = \frac{1}{2\pi} {\sum_{n=0}^{N}}' (-1)^{n+1} n^2 \hat{v}_n,$$
where the prime indicates that the terms $n = 0, N$ are multiplied by $\frac{1}{2}$.

---

These formulas can be explained as follows. The trigonometric interpolant of the extended $\{v_j\}$ data is given by evaluating the inverse FFT at arbitrary $\theta$. Using the $a_n$ coefficients we find that
$$P(\theta) = \frac{1}{2\pi} \sum_{k=-N+1}^{N} e^{ik\theta} \hat{V}_k = \sum_{n=0}^{N} a_n \cos n\theta.$$

The algebraic polynomial interpolant of the $\{v_j\}$ data is $p(x) = P(\theta)$, where $x = \cos\theta$, and the derivative is

$$q'(x) = \frac{Q'(\theta)}{dx/d\theta} = \frac{-\sum_{n=0}^{N} na_n \sin n\theta}{-\sin\theta} = \frac{\sum_{n=0}^{N} na_n \sin n\theta}{\sqrt{1-x^2}}.$$

As for the special formulas for $w_0$ and $w_N$, we determine the value of $q'(x)$ at $x = \pm 1$ by l'Hôpital's rule [Str91], which gives

$$q'(1) = \sum_{n=0}^{N} n^2 a_n, \qquad q'(-1) = \sum_{n=0}^{N} (-1)^{n+1} n^2 a_n.$$

It is straightforward to generalize the method for higher derivatives. At the stage of differentiation in Fourier space we multiply by $(ik)^\nu$ to calculate the $\nu$th derivative, and if $\nu$ is odd, we set $\hat{W}_N = 0$. Secondly, the appropriate factors need to be calculated for converting between derivatives on the equispaced grid and on the Chebyshev grid, i.e., derivatives in the $\theta$ and $x$ variables. For example, the second derivatives are related by

$$q''(x) = \frac{-x}{(1-x^2)^{3/2}} Q'(\theta) + \frac{1}{1-x^2} Q''(\theta). \tag{8.7}$$

If $W_j$ and $W_j^{(2)}$ are the first and second derivatives on the equispaced grid, respectively, then the second derivative on the Chebyshev grid is given by

$$w_j^{(2)} = \frac{-x_j}{(1-x_j^2)^{3/2}} W_j + \frac{1}{1-x_j^2} W_j^{(2)}, \qquad 1 \le j \le N-1.$$

Again, special formulas are needed for $j = 0$ and $N$.

On p. 24 it was mentioned that when the complex FFT is applied to differentiate a real periodic function, a factor of 2 in efficiency is lost. In the method we have just described, the situation is worse, for not only is $V$ real (typically), but it is even (always), and together these facts imply that $\hat{V}$ is real and even too (Exercise 2.2). A factor of 4 is now at stake, and the right way to take advantage of this is to use a *discrete cosine transform (DCT)* instead of an FFT. See [BrHe95], [Van92], and Appendix F of [For96] for a discussion of symmetries in Fourier transforms and how to take advantage of them. At the time of this writing, however, although a DCT code is included in MATLAB's Signal Processing Toolbox, there is no DCT in MATLAB itself. In the following program, `chebfft`, we have accordingly chosen to use the general FFT code and accept the loss of efficiency.

## chebfft.m

```
% CHEBFFT  Chebyshev differentiation via FFT. Simple, not optimal.
%          If v is complex, delete "real" commands.

  function w = chebfft(v)
  N = length(v)-1; if N==0, w=0; return, end
  x = cos((0:N)'*pi/N);
  ii = 0:N-1;
  v = v(:); V = [v; flipud(v(2:N))];      % transform x -> theta
  U = real(fft(V));
  W = real(ifft(1i*[ii 0 1-N:-1]'.*U));
  w = zeros(N+1,1);
  w(2:N) = -W(2:N)./sqrt(1-x(2:N).^2);    % transform theta -> x
  w(1) = sum(ii'.^2.*U(ii+1))/N + .5*N*U(N+1);
  w(N+1) = sum((-1).^(ii+1)'.*ii'.^2.*U(ii+1))/N + ...
              .5*(-1)^(N+1)*N*U(N+1);
```

Program 18 calls `chebfft` to calculate the Chebyshev derivative of $f(x) = e^x\sin(5x)$ for $N = 10$ and 20 using the FFT. The results are given in Output 18. Compare this with Output 11 (p. 56), which illustrates the same calculation implemented using matrices. The differences are just at the level of rounding errors.

To see the method at work for a PDE, consider the wave equation

$$u_{tt} = u_{xx}, \quad -1 < x < 1, \quad t > 0, \quad u(\pm 1) = 0. \tag{8.8}$$

To solve this equation numerically we use a leap frog formula in $t$ and Chebyshev spectral differentiation in $x$. To complete the formulation of the numerical method we need to specify two initial conditions. For the PDE, these would typically be conditions on $u$ and $u_t$. For the finite difference scheme, we need conditions on $u$ at $t = 0$ and at $t = -\Delta t$, the previous time step. Our choice at $t = -\Delta t$ is initial data corresponding to a left-moving Gaussian pulse. Program 19 implements this and should be compared with Program 6 (p. 26). This program, however, runs rather slowly because of the short time step $\Delta t \approx 0.0013$ needed for numerical stability. Time step restrictions are discussed in Chapter 10.

As a second example we consider the wave equation in two space dimensions:

$$u_{tt} = u_{xx} + u_{yy}, \quad -1 < x, y < 1, \quad t > 0, \quad u = 0 \text{ on the boundary}, \tag{8.9}$$

with initial data

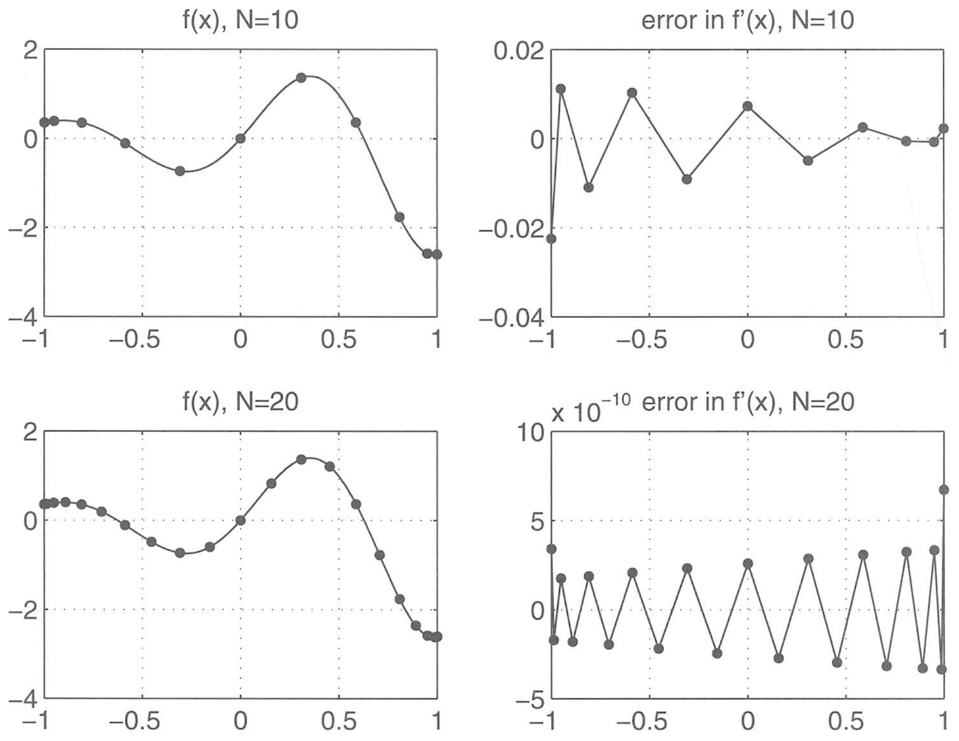
$$u(x, y, 0) = e^{-40((x-0.4)^2+y^2)}, \quad u_t(x, y, 0) = 0.$$

## Program 18

```
% p18.m - Chebyshev differentiation via FFT (compare p11.m)

  xx = -1:.01:1; ff = exp(xx).*sin(5*xx); clf
  for N = [10 20]
    x = cos(pi*(0:N)'/N); f = exp(x).*sin(5*x);
      subplot('position',[.15 .66-.4*(N==20) .31 .28])
      plot(x,f,'.','markersize',14), grid on
      line(xx,ff)
      title(['f(x), N=' int2str(N)])
    error = chebfft(f) - exp(x).*(sin(5*x)+5*cos(5*x));
      subplot('position',[.55 .66-.4*(N==20) .31 .28])
      plot(x,error,'.','markersize',14), grid on
      line(x,error)
      title(['error in f''(x), N=' int2str(N)])
  end
```

## Output 18



Output 18: *Chebyshev differentiation of* $e^x \sin(5x)$ *via FFT. Compare Output* 11 *(p.* 56*), based on matrices.*

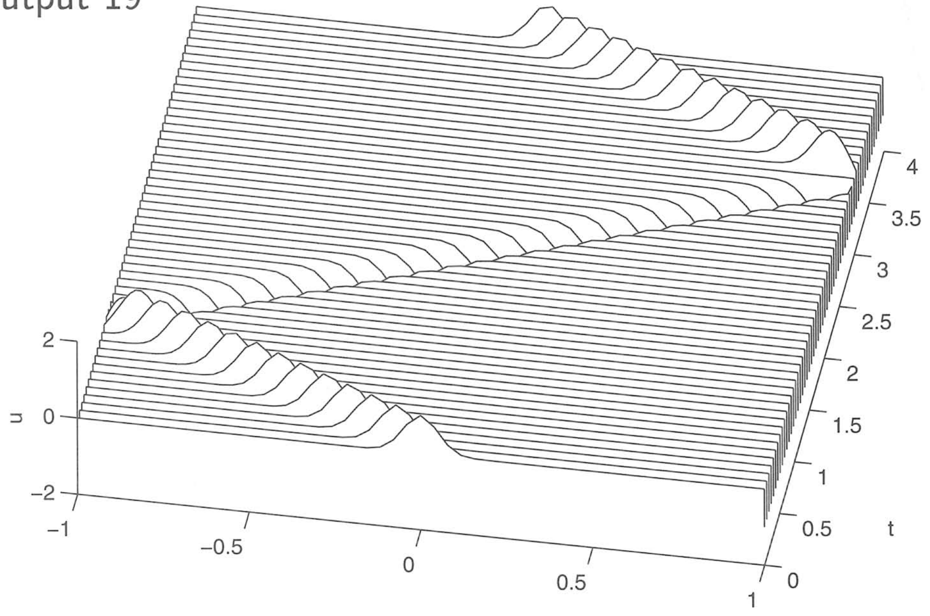
## Program 19

```
% p19.m - 2nd-order wave eq. on Chebyshev grid (compare p6.m)

% Time-stepping by leap frog formula:
  N = 80; x = cos(pi*(0:N)/N); dt = 8/N^2;
  v = exp(-200*x.^2); vold = exp(-200*(x-dt).^2);
  tmax = 4; tplot = .075;
  plotgap = round(tplot/dt); dt = tplot/plotgap;
  nplots = round(tmax/tplot);
  plotdata = [v; zeros(nplots,N+1)]; tdata = 0;
  clf, drawnow, h = waitbar(0,'please wait...');
  for i = 1:nplots, waitbar(i/nplots)
    for n = 1:plotgap
      w = chebfft(chebfft(v))'; w(1) = 0; w(N+1) = 0;
      vnew = 2*v - vold + dt^2*w; vold = v; v = vnew;
    end
    plotdata(i+1,:) = v; tdata = [tdata; dt*i*plotgap];
  end

% Plot results:
  clf, drawnow, waterfall(x,tdata,plotdata)
  axis([-1 1 0 tmax -2 2]), view(10,70), grid off
  colormap([0 0 0]), ylabel t, zlabel u, close(h)
```

## Output 19



Output 19: *Solution of second-order wave equation* (8.8).

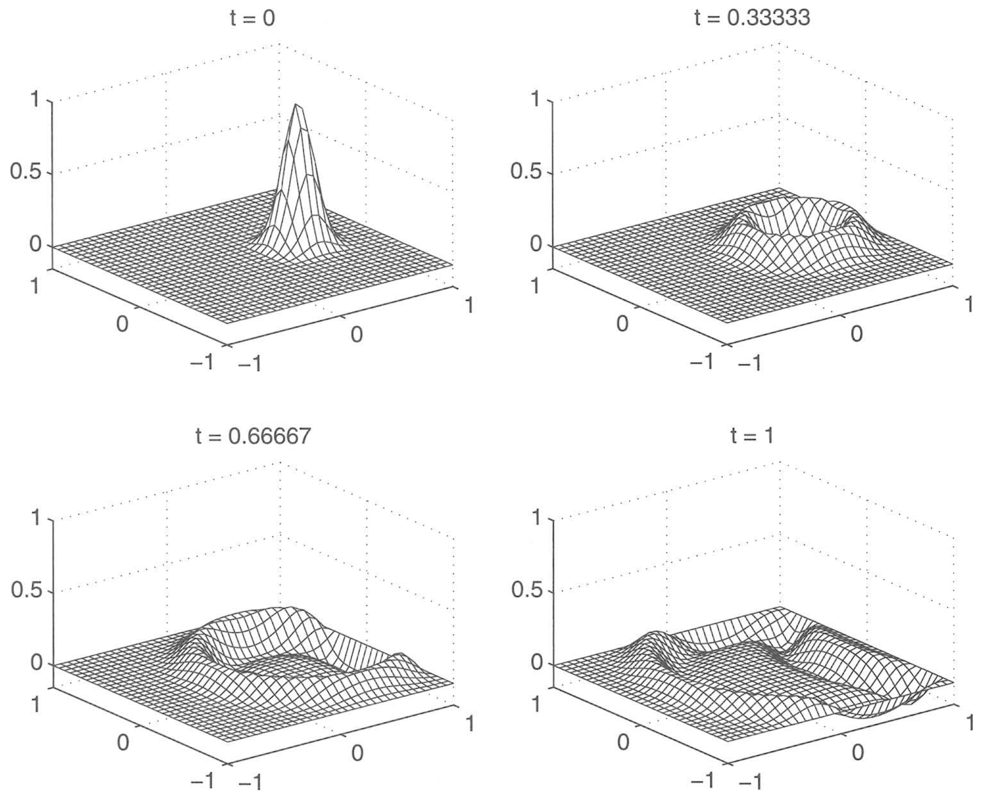
## Program 20

```
% p20.m - 2nd-order wave eq. in 2D via FFT (compare p19.m)

% Grid and initial data:
  N = 24; x = cos(pi*(0:N)/N); y = x';
  dt = 6/N^2;
  [xx,yy] = meshgrid(x,y);
  plotgap = round((1/3)/dt); dt = (1/3)/plotgap;
  vv = exp(-40*((xx-.4).^2 + yy.^2));
  vvold = vv;

% Time-stepping by leap frog formula:
  [ay,ax] = meshgrid([.56 .06],[.1 .55]); clf
  for n = 0:3*plotgap
    t = n*dt;
    if rem(n+.5,plotgap)<1     % plots at multiples of t=1/3
      i = n/plotgap+1;
      subplot('position',[ax(i) ay(i) .36 .36])
      [xxx,yyy] = meshgrid(-1:1/16:1,-1:1/16:1);
      vvv = interp2(xx,yy,vv,xxx,yyy,'cubic');
      mesh(xxx,yyy,vvv), axis([-1 1 -1 1 -0.15 1])
      colormap([0 0 0]), title(['t = ' num2str(t)]), drawnow
    end
    uxx = zeros(N+1,N+1); uyy = zeros(N+1,N+1);
    ii = 2:N;
    for i = 2:N                % 2nd derivs wrt x in each row
      v = vv(i,:); V = [v fliplr(v(ii))];
      U = real(fft(V));
      W1 = real(ifft(1i*[0:N-1 0 1-N:-1].*U)); % diff wrt theta
      W2 = real(ifft(-[0:N 1-N:-1].^2.*U));    % diff^2 wrt theta
      uxx(i,ii) = W2(ii)./(1-x(ii).^2) - x(ii).* ...
                  W1(ii)./(1-x(ii).^2).^(3/2);
    end
    for j = 2:N                % 2nd derivs wrt y in each column
      v = vv(:,j); V = [v; flipud(v(ii))];
      U = real(fft(V));
      W1 = real(ifft(1i*[0:N-1 0 1-N:-1]'.*U));% diff wrt theta
      W2 = real(ifft(-[0:N 1-N:-1]'.^2.*U));   % diff^2 wrt theta
      uyy(ii,j) = W2(ii)./(1-y(ii).^2) - y(ii).* ...
                  W1(ii)./(1-y(ii).^2).^(3/2);
    end
    vvnew = 2*vv - vvold + dt^2*(uxx+uyy);
    vvold = vv; vv = vvnew;
  end
```

## Output 20a

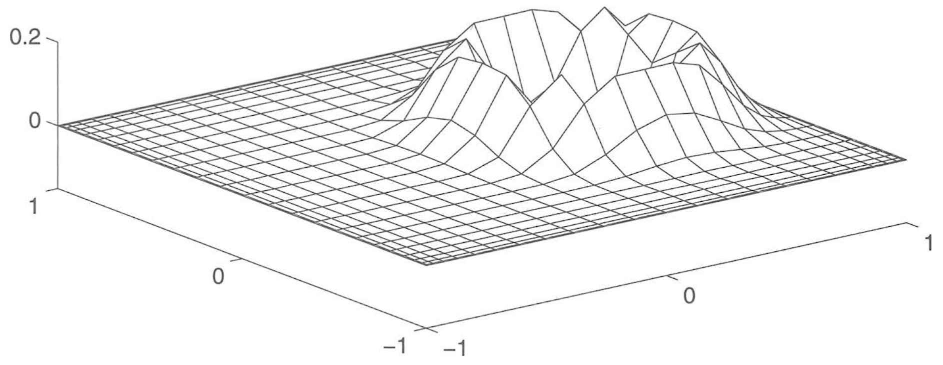


Output 20a: *Solution of the second-order wave equation* (8.9) *on a square.*

Program 20 discretizes this problem with a leap frog formula again for the time discretization and a Chebyshev spectral method on a tensor product grid in $x$ and $y$.

As in Programs 13–17, the solutions of Output 20a have been interpolated to a finer grid than the computational one for a better display. (Global polynomial interpolation would give spectral accuracy, but here, for simplicity, we use MATLAB's local `interp2` command.) They may accordingly give a misleading impression that the computational grid is regular. In fact, it is a spectral grid, clustered at the boundaries, and to emphasize this point, Output 20b shows the raw data at $t = 1/3$ on the spectral grid. This plot was obtained by replacing `mesh(xxx,yyy,vvv)` in Program 20 by `mesh(xx,yy,vv)`. We emphasize that despite the apparent crudity of this plot, the results at each grid point are spectrally accurate, aside from errors of magnitude $O((\Delta t)^2)$ introduced by time-stepping.

## Output 20b



Output 20b: *Same as the second plot of Output* 20, *but without the interpolation to a finer grid.*

**Summary of This Chapter.** Chebyshev differentiation can be carried out by the FFT. The underlying idea is that of transplantation from Chebyshev points on $[-1,1]$ to equally spaced points on the unit circle. Ideally, for real data, a real discrete cosine transform (DCT) should be used, but the general FFT is also applicable with a certain loss of efficiency.

## Exercises

**8.1.** For $n \geq 1$, the leading coefficient of $T_n(x)$ is $c_n = 2^{n-1}$, and this implies $\lim_{n\to\infty}(c_n)^{1/N} = 2$. Derive this limit from the general results of Chapter 5.

**8.2.** Perform a study of the relative efficiencies of `cheb` and `chebfft` as a function of $N$. Do not count the time taken by `cheb` to form $D_N$, just the time taken to multiply $D_N$ by a vector.

**8.3.** Modify Program 12 (p. 57) to make use of `chebfft` instead of `cheb`. The results should be the same as in Output 12, except for rounding errors. Are the effects of rounding errors smaller or larger than before?

**8.4.** Modify Program 20 to make use of matrices instead of the FFT. Make sure to do this elegantly, using matrix-matrix multiplications rather than explicit loops. You will find that the code gets much shorter, and faster, too. How much faster is it? Does increasing $N$ from 24 to 48 tip the balance?

**8.5.** Find a way to modify your program of Exercise 8.4, as in Exercise 3.8 but now for a second-order problem, to make use of matrix exponentials rather than time discretization. What effect does this have on the computation time?

**8.6.** Write a code `chebfft2` for second-order differentiation by the FFT, and show by examples that it matches the results obtained by matrices, apart from rounding errors.

**8.7.** Explain how the entries of the Chebyshev differentiation matrix $D_N$ could be computed by suitable calls to `chebfft` rather than by explicit formulas as in Theorem 7 (p. 53). What is the asymptotic operation count for this method as $N \to \infty$?

**8.8.** Write a code `chebdct` that computes the same result as `chebfft`, but makes use of the function DCT from MATLAB's Signal Processing Toolbox. (The code will be restricted to real data.) What gain of efficiency do you observe?

**8.9.** Suppose $p(x) = \sum_{n=0}^{N} a_n T_n(x) = \sum_{n=0}^{N} c_n x^n$, and consider the vectors $a = (a_0, \dots, a_N)^T$ and $c = (c_0, \dots, c_N)^T$. Let $A$ be the $(N+1) \times (N+1)$ matrix such that $c = Aa$. Write down $A$ in the case $N = 3$. Now write a short and elegant MATLAB function `cheb2mon` such that the command `A = cheb2mon(N)` constructs this matrix $A$. Use `cheb2mon` to determine what polynomial $\sum_{n=0}^{5} c_n x^n$ corresponds to $T_0(x) - 2T_1(x) + 3T_2(x) + 2T_3(x) + T_4(x) - T_5(x)$ and what polynomial $\sum_{n=0}^{5} a_n T_n(x)$ corresponds to $1 - 2x + 3x^2 + 2x^3 + x^4 - x^5$.