ELSEVIER

Applied Numerical Mathematics 34 (2000) 275–292

APPLIED NUMERICAL MATHEMATICS

www.elsevier.nl/locate/apnum

Spectral methods in computing invariant tori ☆

Manfred R. Trummer [1]

[a] *Department of Mathematics, Simon Fraser University, Burnaby, British Columbia V5A 1S6, Canada*

Abstract

We present two new spectral implementations for computing invariant tori. The underlying nonlinear partial differential equation (Dieci et al., 1991), although hyperbolic by nature, has periodic boundary conditions in both space and time. Our first approach uses a spatial spectral discretization, and finds the solution via a shooting method. The second one employs a full two-dimensional Fourier spectral discretization, and uses Newton's method. This leads to very large, sparse, unsymmetric systems, although with highly structured matrices. A modified conjugate gradient type iterative solver was found to perform best when the dimensions get too large for direct solvers. The two methods are implemented for the van der Pol oscillator, and compared to previous algorithms.
© 2000 IMACS. Published by Elsevier Science B.V. All rights reserved.

Keywords: Dynamical systems; Nonlinear ODEs; Invariant torus; Pseudospectral method; Iterative methods for nonsymmetric systems

1. Introduction

Computational aspects of dynamical systems are becoming more important as more nonlinear mathematical models in science are being studied. To understand these models one often needs to employ analytical, geometric and computational techniques. For ordinary differential equations (ODEs), invariant manifolds are a key feature to look for. Even though the dynamics of the ODE can be very complicated, such invariant manifolds need not be. In this paper we will concentrate on computing one of the simpler types of such manifolds, namely tori. Our methods are based on solving an associated partial differential equation (PDE); to our knowledge, this approach was first exploited computationally in [6].

Let us consider the autonomous first-order system of ODEs

$$\frac{\mathrm{d}w}{\mathrm{d}t} = F(w), \quad w \in W \subset \mathbb{R}^n, \tag{1.1}$$

☆ This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) grant OGP0036901, NSERC and Schweizerischer Nationalfonds zur Förderung der Wissenschaften BEF 0150297, Forschungsinstitut für Mathematik, ETH Zürich and the Department of Mathematics at the University of Auckland.

[1] E-mail: trummer@sfu.ca

0168-9274/00/$20.00 © 2000 IMACS. Published by Elsevier Science B.V. All rights reserved.
PII: S0168-9274(99)00133-6

where F is a smooth mapping from W to $\mathbb{R}^n$. We assume that a smooth invariant manifold $\Omega \in W$ exists, i.e., for any initial data on Ω the solution of (1.1) will remain on the manifold Ω. Our aim is to numerically approximate Ω. Following [6], we restrict ourselves to systems (1.1) of the form

$$\theta_t = f(\theta, r), \qquad r_t = g(\theta, r), \tag{1.2}$$

with $\theta \in U \subset \mathbb{R}^p$, $r \in V \subset \mathbb{R}^q$, and we make the critical assumption that the manifold Ω can be written as $\{(\theta, \Phi(\theta))\colon\ \theta \in U\}$, where $\Phi : U \to V$, i.e., we assume that Ω can be parameterized over U. This is a severe restriction, even though in principle such a splitting is always possible *locally* via the implicit function theorem. The aim of this paper, however, is to contrast our numerical methods to previous ones. To compute Φ, i.e. Ω, one has to solve the system of nonlinear PDEs

$$\boldsymbol{J}_{\Phi}(\theta) f\big(\theta, \Phi(\theta)\big) = g\big(\theta, \Phi(\theta)\big), \quad \theta \in U, \tag{1.3}$$

where $\boldsymbol{J}_{\Phi}$ denotes the Jacobian matrix of Φ, with appropriate boundary conditions (see [6]). Note that this procedure can be seen as a natural extension of phase space analysis for systems of ODEs in $\mathbb{R}^2$.

Here we assume that U is a torus, a manifold without boundary. Therefore, the boundary conditions for (1.3) are periodic in each component of r.

We denote by

$$T^p := \big\{\theta = (\theta_1, \ldots, \theta_p)\colon\ \theta_j \in \mathbb{R} \bmod 2\pi\big\} \tag{1.4}$$

the p-dimensional torus. We will briefly rewrite the PDE (1.3) for the case where U can be identified with T^2, using t and x as symbols for the independent variables θ_1 and θ_2, and where $V = \mathbb{R}$, i.e., the function r is a scalar. This will put our notation in line with a more familiar "PDE setting". Eq. (1.3) simply becomes

$$f_1(t, x, r) r_t + f_2(t, x, r) r_x = g(t, x, r), \quad (t, x) \in [0, 2\pi] \times [0, 2\pi], \tag{1.5}$$

$$r(0, x) = r(2\pi, x), \qquad r(t, 0) = r(t, 2\pi). \tag{1.6}$$

If we can divide by f_1 in Eq. (1.5) we finally obtain

$$r_t + \phi(t, x, r) r_x - \psi(t, x, r) = 0, \tag{1.7}$$

where

$$\phi(t, x, r) := \frac{f_2(t, x, r)}{f_1(t, x, r)}, \qquad \psi(t, x, r) := \frac{g(t, x, r)}{f_1(t, x, r)}. \tag{1.8}$$

In the following we introduce two methods based on spectral discretizations to solve this problem. We refer the reader to [3–6] for extensive discussions on various other discretizations and their properties. In particular, there is concern about the performance of methods for values of the continuation parameter λ near "torus breakdown", as the solution loses smoothness. This is more of a challenge for spectral methods than for lower order discretizations. A comprehensive discussion of the use of iterative methods for solving the linear systems arising in Section 3 can be found in [12].

2. Shooting type approach

In our first approach, we use a Fourier collocation (or pseudospectral) method to discretize in "space" (i.e., with respect to the x variable), and then use a shooting type approach to determine the correct initial

values (i.e., the ones which give rise to 2π periodicity in time). For the time integration we use a simple, explicit linear multistep method, for example, an Adams–Bashforth method.

We partition the interval $[0, 2\pi]$ into N subintervals of equal length (for simplicity we assume N to be a power of 2), and approximate $r(t, j2\pi/N)$ by u_j, $j = 0, 1, \ldots, N-1$. The partial derivative r_x is computed by the Fourier collocation method (see, e.g., [2]). Numerically this can be achieved by a simple matrix–vector multiplication, or, by a fast Fourier transform (FFT), a multiplication by a diagonal matrix, and an inverse FFT. Here we choose the latter, as in a straightforward implementation the FFT approach often results in somewhat higher accuracy (see, e.g., [8]). This set-up automatically takes care of the periodicity conditions in x. Given initial conditions $\boldsymbol{u} := \boldsymbol{u}(0) := (u_0(0), \ldots, u_{N-1}(0))^{\mathrm{T}}$, we numerically integrate Eq. (1.5) from $t = 0$ to $t = 2\pi$ giving us $\boldsymbol{v} := \boldsymbol{u}(2\pi)$. Thus we have a nonlinear operator M defined on $\mathbb{R}^N$ by

$$\boldsymbol{u}(2\pi) = \boldsymbol{v} = M(\boldsymbol{u}) = M\big(\boldsymbol{u}(0)\big). \tag{2.1}$$

To find an approximate solution of (1.5) satisfying both boundary conditions (1.6) we simply need to find a fixed point of the nonlinear operator M, i.e., a vector $\boldsymbol{v}$ satisfying

$$\boldsymbol{v} = M(\boldsymbol{v}). \tag{2.2}$$

One could try to use the method of successive iterations to find solutions of (2.2). There is no reason, however, to expect the Jacobian matrix of M to have a spectral radius less than unity to guarantee convergence of the ordinary iteration method. Therefore, we resort to a secant method, namely Broyden's method (see [7]) to solve (2.2). Given an initial guess $\boldsymbol{v}_0$ for $\boldsymbol{v}$, and an initial guess A_0 for the Jacobian matrix of M, the algorithm is as follows:

$$\begin{aligned}
&\boldsymbol{s}_k := -A_k^{-1} M(\boldsymbol{v}_k),\\
&\boldsymbol{v}_{k+1} := \boldsymbol{v}_k + \boldsymbol{s}_k,\\
&\boldsymbol{y}_k := M(\boldsymbol{v}_{k+1}) - M(\boldsymbol{v}_k),\\
&A_{k+1} := A_k + \frac{(\boldsymbol{y}_k - A_k \boldsymbol{s}_k)\boldsymbol{s}_k^{\mathrm{T}}}{\boldsymbol{s}_k^{\mathrm{T}} \boldsymbol{s}_k}.
\end{aligned} \tag{2.3}$$

The initial guess for A_0 can be taken as a finite difference approximation to the Jacobian of M at $\boldsymbol{v}_0$. Although this is relatively expensive, our experience shows that the better performance of algorithm (2.3) with a good initial guess justifies the cost. It should be noted that in situations where one applies continuation, both initial guesses are readily available. One of the major drawbacks of this approach is its sensitivity to the stability of the time integration (or lack thereof), even when good initial guesses are available. This is not unlike the problems one encounters with shooting methods for two-point boundary value problems (see, e.g., [1]). Implicit time stepping may alleviate this problem but at considerable cost. On a more fundamental level, the different treatment afforded to the x and t variable seems unjustified in our problem.

3. The full spectral scheme

In the previous section we described a scheme which is spectral only in one variable (in "space"). Now we discretize spectrally also along the "time" variable t. The beauty of this approach lies in the ease of treatment of our periodic boundary conditions—they are implicitly taken care of by the

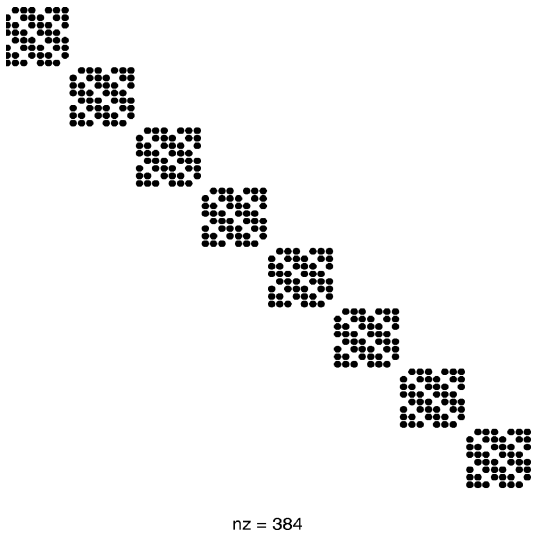

Fig. 1. Sparsity structure of $\mathrm{D}^{[x]}$.

Fourier collocation approximation. Also, as we shall see, we discretize the full nonlinear problem; an expression for the Jacobian matrix is readily available, and thus Newton's method solves the nonlinear problem. The major work is then concentrated in computing the Newton step, which requires the solution of an N^2 by N^2 system of linear equations, where N is the number of discretization points in each of the two directions. The matrix of this system is sparse; it has $\mathrm{O}(N^3)$ nonzero elements, with full nonzero N by N blocks on the diagonal, and nonzero elements on every Nth subdiagonal. When this system becomes too large to be solved efficiently by direct methods, we switch to an iterative solver. The matrix–vector multiply can be performed quite efficiently due to the special structure of the matrix, and only very little storage space is needed. A potential drawback of the spectral approach is a loss of efficiency if the solution is not smooth. In the context of the torus problem this occurs near "breakdown" of the torus, a phenomenon of interest. Our experiments indicate a deterioration of the performance of our method in this scenario, but reasonable results are still obtained, in particular, away from the discontinuities. A more detailed study following the route to breakdown will be reported elsewhere.

The differentiation in both the x and t direction is done spectrally, independently of each other. We denote the respective spectral differentiation matrices by $\mathrm{D}^{[x]}$, and $\mathrm{D}^{[t]}$. Each one of these two matrices can be obtained by a permutation of rows and columns of the other one. The difference in their structure comes from how the two-dimensional array of approximate solutions, $\boldsymbol{u} = (u_{j,k})$, $u_{j,k} \approx r(t_j, x_k)$, is arranged into the one-dimensional vector $\boldsymbol{v} \in \mathbb{R}^{N^2}$. We set

$$\boldsymbol{v} := (u_{1,1}, u_{1,2}, \ldots, u_{1,N}, u_{2,1}, u_{2,2}, \ldots, u_{2,N}, u_{3,1}, u_{3,2}, \ldots)^{\mathrm{T}}, \tag{3.1}$$

i.e., we write the elements of $\boldsymbol{u}$ row-by-row into the vector $\boldsymbol{v}$. Therefore, we have all the discrete values of $r(t, x)$ for fixed values of the time variable t in contiguous storage locations in $\boldsymbol{v}$, which implies that the matrix $\mathrm{D}^{[x]}$ (corresponding to spectral differentiation with respect to the space variable x) is a

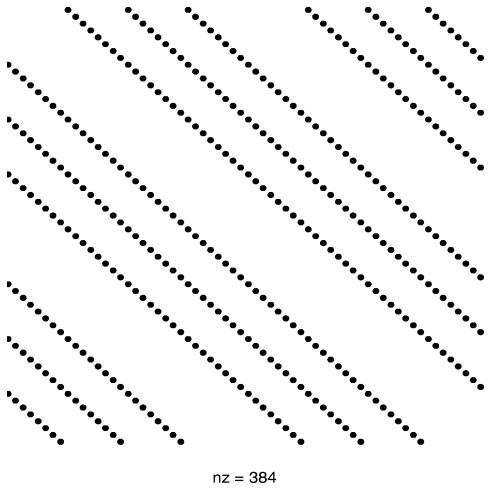

Fig. 2. Sparsity structure of $\mathrm{D}^{[t]}$.

block-diagonal matrix with N constant N by N diagonal blocks. For N even, each diagonal block is the skew-symmetric Toeplitz matrix

$$\boldsymbol{\Lambda} = \frac{1}{2}\begin{bmatrix} 0 & +\cot\left(\frac{\pi}{N}\right) & -\cot\left(\frac{2\pi}{N}\right) & +\cot\left(\frac{3\pi}{N}\right) & \cdots \\ -\cot\left(\frac{\pi}{N}\right) & 0 & +\cot\left(\frac{\pi}{N}\right) & -\cot\left(\frac{2\pi}{N}\right) & \ddots \\ +\cot\left(\frac{2\pi}{N}\right) & -\cot\left(\frac{\pi}{N}\right) & 0 & +\cot\left(\frac{\pi}{N}\right) & \ddots \\ \vdots & +\cot\left(\frac{2\pi}{N}\right) & -\cot\left(\frac{\pi}{N}\right) & 0 & \ddots \\ \vdots & \ddots & & & \ddots \end{bmatrix}. \tag{3.2}$$

The matrix $\mathrm{D}^{[t]}$ is an $N^2 \times N^2$ sparse Toeplitz matrix, with at most every Nth diagonal being nonzero. Figs. 1 and 2 show the sparsity structure of these spectral differentiation matrices for $N = 8$. Note, that the number of nonzero elements is $\mathrm{O}(N^3)$. Multiplications of $\mathrm{D}^{[x]}$ or $\mathrm{D}^{[t]}$ by a vector require only the $N \times N$ matrix $\boldsymbol{\Lambda}$.

Recall Eq. (1.7); its discretized form is simply

$$\mathrm{D}^{[t]}\boldsymbol{v} + \mathrm{diag}\left(\vec{\phi}(\boldsymbol{v})\right)\left(\mathrm{D}^{[x]}\boldsymbol{v}\right) - \vec{\psi}(\boldsymbol{v}) = 0, \tag{3.3}$$

where $\vec{\phi}(\boldsymbol{v})$ and $\vec{\psi}(\boldsymbol{v})$ denote vectors with entries $\phi(t_i, x_j, u_{i,j})$ and $\psi(t_i, x_j, u_{i,j})$, respectively, ordered in the same way as the $u_{i,j}$ in Eq. (3.1). As usual $\mathrm{diag}(\vec{\phi}(\boldsymbol{v}))$ denotes the diagonal matrix with the vector $\vec{\phi}(\boldsymbol{v})$ as its diagonal.

To solve Eq. (3.3) for $\boldsymbol{v}$, we will use an initial guess, and then apply Newton's method. The corresponding Jacobian matrix is

$$J = \mathrm{D}^{[t]} + \mathrm{diag}\left(\vec{\phi}(\boldsymbol{v})\right)\mathrm{D}^{[x]} + \boldsymbol{\Phi}_v\,\mathrm{diag}\left(\mathrm{D}^{[x]}\boldsymbol{v}\right) - \boldsymbol{\Psi}_v, \tag{3.4}$$

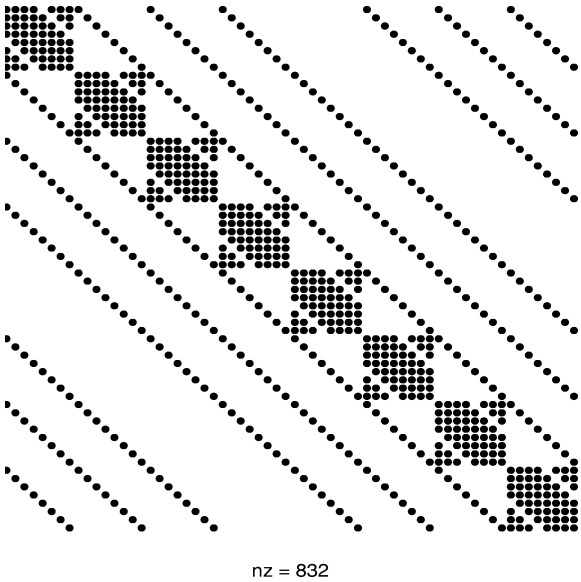

Fig. 3. Sparsity structure of the Jacobian matrix in Eq. (3.4).

where Φ_v and Ψ_v are diagonal matrices with diagonal elements

$$\frac{\partial\phi}{\partial r}(t_i, x_j, u_{i,j}) \quad \text{and} \quad \frac{\partial\psi}{\partial r}(t_i, x_j, u_{i,j}),$$

respectively, again ordered as in Eq. (3.1). Here, $\partial/\partial r$ denotes the partial derivative with respect to the third argument, the unknown function r.

Fig. 3 shows the sparsity structure of the Jacobian matrix J. It has also $O(N^3)$ nonzero elements. However, to multiply a vector by J again only the matrix $\boldsymbol{\Lambda}$ is needed, i.e., the storage requirement is only [2] $O(N)$—or even less, if one is willing to recompute the entries of the matrix Λ when needed.

4. The iterative solver: BiCGStab2 and BiCGstab(ℓ)

In this section we will give a brief description of variants of the *biconjugate gradient method* (see [9,13]) for solving nonsymmetric linear systems $Az = b$, where A is a nonsingular M by M matrix. The basic idea of the biconjugate gradient method is to generate from an initial approximation z_0 a sequence (z_n) such that the nth residual $r_n := b - Az_n$ lies in the Krylov space generated by A and r_0,

$$\mathcal{K}_{n+1} := \operatorname{span}\{r_0, Ar_0, \ldots, A^n r_0\}, \tag{4.1}$$

and is orthogonal to the Krylov space

$$\mathcal{L}_n := \operatorname{span}\{y_0, A^{\mathrm{T}} y_0, \ldots, (A^{\mathrm{T}})^{n-1} y_0\} \tag{4.2}$$

[2] Since N is only of moderate size, it appears to be advantageous to store all of the matrix entries of $\boldsymbol{\Lambda}$, requiring N^2 storage locations.

generated from some initial vector y_0 and by A^{T}, the transpose of A (in the complex case, replace the transpose by the Hermitian). It can then be shown that the residuals r_n satisfy

$$r_n = \rho_n(A)r_0, \tag{4.3}$$

where (ρ_n) is a sequence of orthogonal polynomials, normalized by $\rho_n(0) = 1$. For symmetric A, and $y_0 = r_0$, this method becomes the classical conjugate gradient method. Loosely speaking, the convergence properties of these conjugate gradient algorithms depend on how small the polynomials ρ_n are on the spectrum of A. Since the spectrum of A consists of at most M distinct points in the complex plane, one can hope that $\rho_M(\lambda) = 0$ for all eigenvalues λ (the normalization $\rho_M(0) = 1$ provides the $(M + 1)$st interpolation condition), i.e., under the absence of round-off errors the algorithm terminates no later than at the Mth step with the exact solution $z = z_M$. This property is well known to hold for the classical Lanczos method, and it is also true for the biconjugate gradient algorithm. However, in practice orthogonality can often not be maintained, and what really counts is the good convergence behaviour (and the small memory requirements) of the respective iterative method.

One obvious choice for the iterate z_n is to minimize the residual r_n over the space $z_0 + \mathcal{K}_n$. This gives rise to the GMRES algorithm terminating in no more than M iterations, which is optimal for the Krylov subspaces in question, but no longer has short recurrences. To find r_n a linear least squares problem of size $n + 1$ by n must be solved, making GMRES exceedingly expensive as the iteration progresses. In GMRES(k) we restart the GMRES algorithm after k iterations. Although like in GMRES the convergence behaviour is still monotone, optimality is lost and finite termination is no longer guaranteed. Another straightforward way of dealing with nonsymmetric systems is to apply standard conjugate gradient to the symmetric system $A^{\mathrm{T}}Az = A^{\mathrm{T}}b$ or $AA^{\mathrm{T}}y = b$. The former results in the CGNR algorithm (minimizing the residuals $r_n = b - Az_n$), whereas the latter is CGNE (minimizing the errors $z_n - A^{-1}b$). Both of these algorithms require multiplication by the transpose of the original matrix A. Their performance depends on the singular values of A. We will compare the biconjugate gradient algorithms to GMRES(20) and CGNR. In our experiments CGNR consistently outperformed CGNE, and GMRES became too expensive.

Conditions (4.1) and (4.2) lead to recurrence formulas for the vectors z_n; more precisely, two pairs of finite vector sequences are generated, (x_n), (y_n), and (u_n), (v_n), satisfying the biorthogonality conditions

$$y_n^{\mathrm{T}}x_m = 0, \quad v_n^{\mathrm{T}}Au_m = 0, \qquad n \neq m. \tag{4.4}$$

It is one of the important features of the Lanczos approach that observing the orthogonality condition (4.4) for $m = n + 1$ implies that they hold for all $m \leqslant n$. This allows for short recurrences. However, in the presence of round-off errors the biorthogonality is often lost. Moreover, the convergence properties for Lanczos types methods for unsymmetric problems are not very well understood. Indeed, the convergence behaviour can be quite erratic, and is far from monotone.

One of the disadvantages of the unsymmetric Lanczos method is that in addition to computing matrix–vector products with A we must compute products with A^{T}. Sonneveld's CGS method [17] addresses this problem, requiring two multiplications by A in each step of the iteration instead. The residuals in the CGS algorithm satisfy

$$r_n = {\rho_n}^2(A)r_0, \tag{4.5}$$

where the ρ_n are the same polynomials as in (4.3). Van der Vorst [20] then derived another method, called BiCGStab, from CGS, by computing residuals of the form

$$r_n = (\rho_n\tau_n)(A)r_0, \tag{4.6}$$

where the ρ_n are again the Lanczos polynomials of (4.3), and the τ_n are polynomials of degree n, with $\tau_n(\zeta) = \tau_{n-1}(\zeta)(1 - \chi_{n-1}\zeta)$, and in each recurrence step the additional new root of τ_n is chosen to minimize $\|r_n\|$, subject to the new root being real.

Gutknecht [11] then argued that for nonsymmetric matrices (even real ones), the spectrum is often not real, and to achieve good damping, one should allow the τ_n to have complex roots (the success of the algorithm depends now on the size of $\rho_n(\lambda)\tau_n(\lambda)$ for λ in the spectrum of A—or to be more precise, for λ in the pseudospectrum of A; see, e.g., [18]). To keep the arithmetic real for real matrices, these complex roots are introduced in complex conjugate pairs, and again are chosen to minimize the new residual (more precisely, at every odd step one extra real root is introduced which in turn is removed in the following (even) step, when possibly a pair of complex conjugate roots is introduced). In Gutknecht's version BiCGStab2, this minimization takes place over a two-dimensional subspace, and is carried out via an orthogonal projection. The resulting algorithm can be found in [11]. Gutknecht's algorithm also avoids multiplications by A^{T}, and requires two multiplications by A in each step.

Sleijpen and Fokkema [16] introduced BiCGstab(ℓ), which generalizes Gutknecht's algorithm. For $\ell = 1$ this algorithm is equivalent to BiCGStab; in exact arithmetic BiCGstab(2) produces the same results as BiCGStab2 as long as the latter does not break down. In particular, the new algorithm BiCGstab(2) avoids the introduction of the extra real root at every odd step. It was pointed out in [16] that the odd steps in BiCGStab2 can create the problem of a potentially nearly degenerate minimum residual polynomial being introduced, and the large errors created thereby may severely pollute the Bi-CG iteration coefficients, and slow down the whole process dramatically. For $\ell \geqslant 2$ BiCGstab(ℓ) takes ℓ BiCG steps, and then performs a minimization over an ℓ-dimensional subspace. Each step requires now 2ℓ matrix–vector [3] products. For larger ℓ the minimization procedures become more expensive, and because of the modified Gram–Schmidt orthonormalization method there is also a potential for numerical instability. However, moderate values of ℓ appear to be producing good results.

For more details on BiCGstab(ℓ) the reader is referred to [16]. Here we include a MATLAB implementation of the algorithm. It should be noted that in our program the explicit multiplication by A ought be replaced by a subroutine call which returns the matrix–vector product. This is what we did in our numerical experiments which were performed with CGS, GMRES(20), BiCGStab2, BiCGstab(ℓ) and CGNR, the classical conjugate gradient algorithm applied to the normal equations. Clearly, BiCGStab(ℓ) gives the most reliable results, and the choice $\ell = 8$ appears to be the best one.

5. Numerical results

In this section we will report briefly our numerical results for the forced Van der Pol oscillator (see, e.g., [21]). The corresponding equation is

$$\ddot{x} + \alpha\left(x^2 - 1\right)\dot{x} + x = \beta\cos(\omega t). \tag{5.1}$$

For positive α a numerical study of this equation is contained in [21], for several values of the parameters $k = \beta/2\alpha$ and $\sigma = (1 - \omega^2)/\alpha$. In [6] the values $\alpha = k = \sigma = 0.4$ are used, and those are the parameters

[3] One step of BiCGstab(2) is equivalent to two steps in BiCGStab2.

Algorithm BiCGstab(ℓ)

```
function [xs,res] = bcgstabl(ell,A,r,x,y,nit,tol);
%BCGSTABL Conjugate Gradient (BICGSTAB(L) by Sleijpen & Fokkema)
%         for nonsymmetric systems Ax = b
%
% [xs,res] = bcgstabl(ell,A,r,x,y,nit,tol)
%
% INPUT:     ell   parameter L >=2
%            A     matrix of system
%            r     initial residual b - Ax
%            x     initial vector
%            y     generating vector for "L-Krylov space"
%            nit   maximum number of iterations
%            tol   error tolerance
% OUTPUT:    xs    Final approximation
%            res   residual norm of each iterate (optional)
%
N=length(r); Shift=diag(ones(1,ell-1),1); it=0; xs=x;
gamma=zeros(ell,1); gammap=gamma; gamma2p=gamma; sigma=gamma;
uhat=zeros(N,ell+1); rhat=uhat; rhat(:,1)=r;
rho0=1; alfa=0; omega=1;
while ((norm(r) > tol) & (it < nit)),
  it=it+1; rho0=-omega*rho0; T=eye(ell,ell);
  for j=1:ell,                                                    %%%%
    rho1=y'*rhat(:,j); beta=alfa*rho1/rho0; rho0=rho1;              %
    for i=1:j, uhat(:,i)=rhat(:,i)-beta*uhat(:,i);  end; %for i   % Bi-CG
    uhat(:,j+1)=A*uhat(:,j); gam=y'*uhat(:,j+1); alfa=rho0/gam;   % part
    for i=1:j, rhat(:,i)=rhat(:,i)-alfa*uhat(:,i+1); end; %for i  %
    rhat(:,j+1)=A*rhat(:,j); xs = xs + alfa*uhat(:,1); %
  end; %for j                                                     %%%%
  for j=2:ell+1,
    for i=2:j-1,                                                  %
      T(i-1,j-1) = rhat(:,j)'*rhat(:,i)/sigma(i-1);               %
      rhat(:,j) = rhat(:,j) - T(i-1,j-1)*rhat(:,i);               % MR
    end; %for i                                                   % part
    sigma(j-1)=rhat(:,j)'*rhat(:,j);                              %
    gammap(j-1) = rhat(:,1)'*rhat(:,j)/sigma(j-1);                %
  end; %for j                                                     %
  omega=gammap(ell); gamma=T\gammap; gamma2p = T*Shift*gamma; %%%%
  xs = xs + rhat(:,1:ell)*[gamma(1) gamma2p(1:ell-1)']';      %%%%
  rhat(:,1) = rhat(:,1) - rhat(:,2:ell+1)*gammap;                 % Update
  uhat(:,1) = uhat(:,1) - uhat(:,2:ell+1)*gamma;              %%%%
  r=rhat(:,1); if (nargout>=2), res(it)=norm(r); end; %if
end; %while %END bicgstabl
```

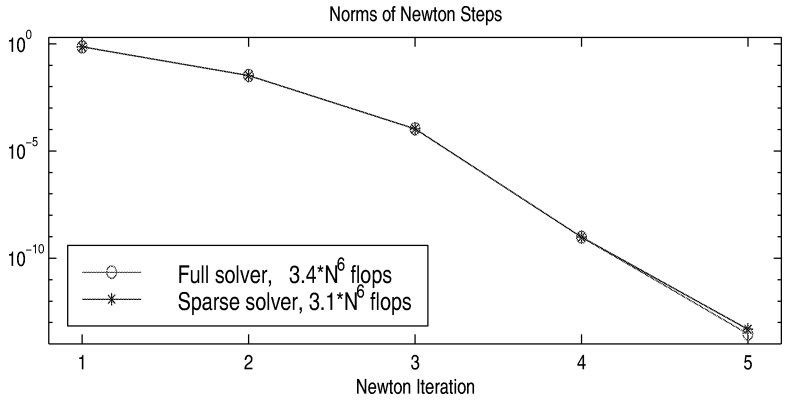

Fig. 4. Convergence of Newton's method, $N = 32$.

in our experiments as well. Following [10] we rewrite the ODE (5.1) as the first-order system

$$\begin{aligned} \dot{x} &= y - \alpha p(x), \\ \dot{y} &= -x + \beta \cos(\omega t), \end{aligned} \tag{5.2}$$

where

$$p(x) = \frac{x^3}{3} - x.$$

Setting $\theta_1 := \omega t$, and rewriting (5.2) in polar coordinates $x = r\cos\theta_2$, $y = r\sin\theta_2$, we finally obtain the autonomous system of ordinary differential equations

$$\begin{aligned} \dot{\theta_1} &= \omega =: f_1(\theta_1, \theta_2, r), \\ \dot{\theta_2} &= -1 + \frac{1}{r}\big(\alpha p(r\cos\theta_2)\sin\theta_2 + \beta\cos\theta_2\cos\theta_1\big) =: f_2(\theta_1, \theta_2, r), \\ \dot{r} &= -\alpha p(r\cos\theta_2)\cos\theta_2 + \beta\sin\theta_2\cos\theta_1 =: g(\theta_1, \theta_2, r). \end{aligned} \tag{5.3}$$

This is now exactly in the form of Eq. (1.2), leading as in Section 1 to the scalar PDE (1.5), (1.6), with f and g defined in (5.3).

At first we present some results on the performance of the full spectral scheme with Newton iteration and direct linear solvers. One must bear in mind that extremely fine discretizations will result in systems too large to be handled by a direct matrix solver. With N collocation points in each dimension, the resulting matrices are $N^2 \times N^2$ with $O(N^3)$ nonzero elements. However, with spectral methods fairly few collocation points will lead to accurate solutions as long as they are reasonably smooth. Fig. 4 shows the norms of the Newton steps in a computation with $N = 32$. As a starting guess for the iteration we used the numerical solution for $N = 16$ and interpolated it bilinearly onto the 32×32 grid. The results demonstrate clearly the quadratic rate of convergence of Newton's method. Fig. 5 shows a picture of the solution for $N = 32$. In all these computations we used the parameter values $k = 0.4$, $\alpha = \sigma = k$. For these parameter values the solution is very smooth.

With $N = 32$ one is already pushing the computational limits on workstations when a full matrix solver is used. Unfortunately, a sparse direct solver is not very promising either, as the matrix J of Eq. (3.4) (see Fig. 3), although sparse, has almost full bandwidth, which will inevitably result in substantial fill-in

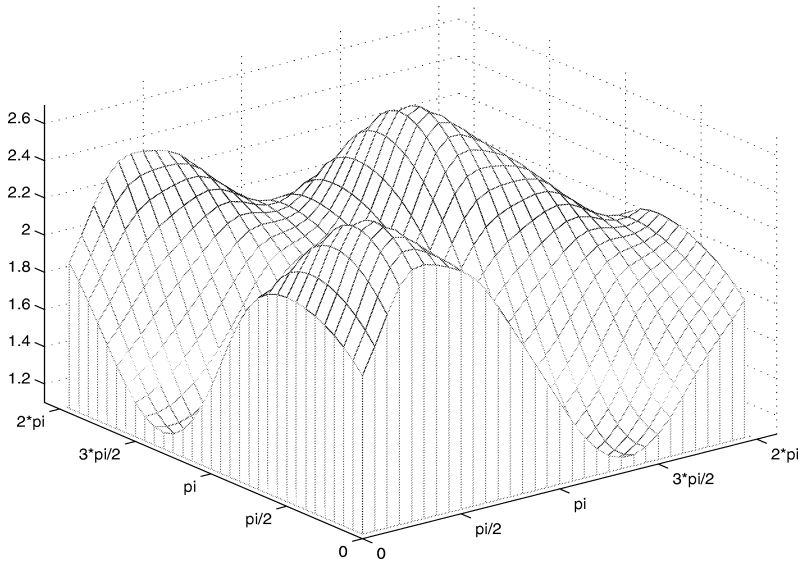

Fig. 5. Van der Pol oscillator, invariant torus, $N = 32$.

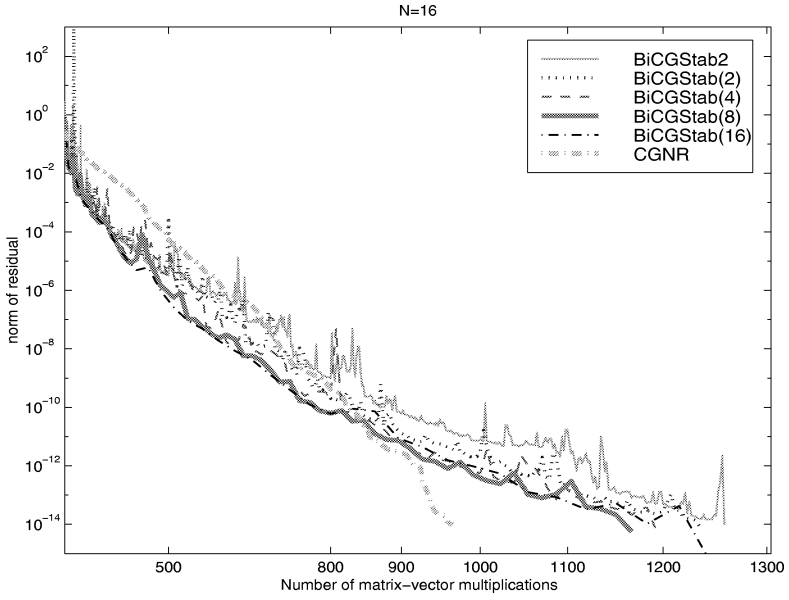

Fig. 6. Errors in BiCGStab2, BiCGStab(ℓ) and CGNR iteration, $N = 16$.

during the solution process. Even the usual reordering algorithms produce almost full L and U factors, when Gaussian elimination is applied to our matrix. Indeed, the cost savings of a sparse solver compared to a full solver are only about 10%.

Thus, iterative methods offer themselves as an attractive alternative. Figs. 6–8 show the results for typical cases with $N = 16$, $N = 32$ and $N = 64$ (the corresponding systems are of size 256×256,

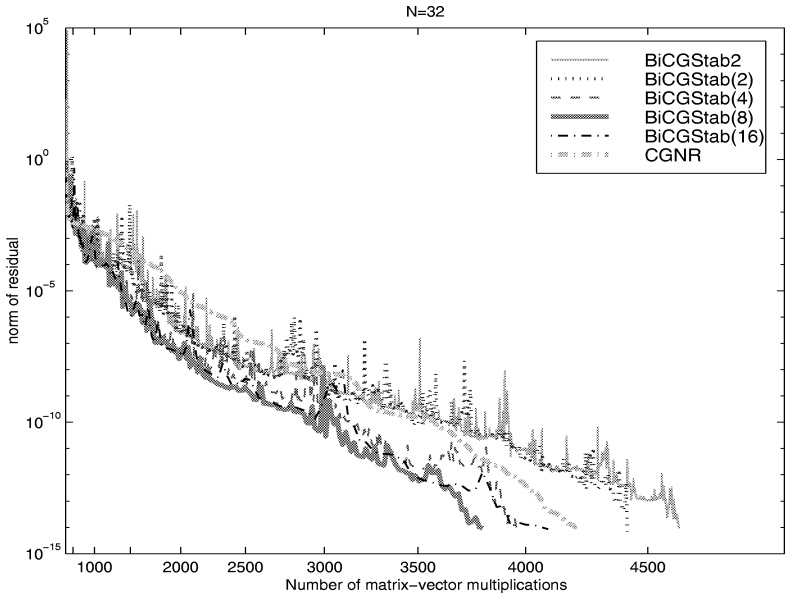

Fig. 7. Errors in BiCGStab2, BiCGStab(ℓ) and CGNR iteration, $N = 32$.

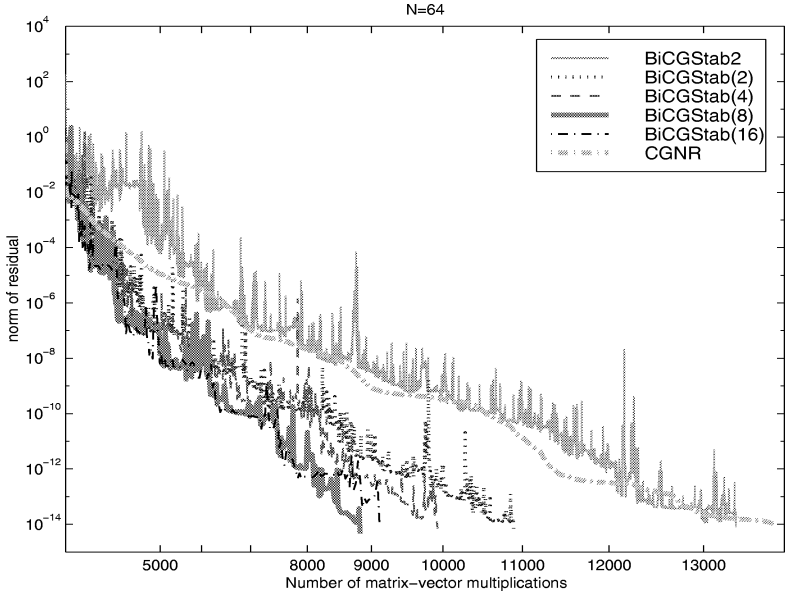

Fig. 8. Errors in BiCGStab2, BiCGStab(ℓ) and CGNR iteration, $N = 64$.

1024×1024 and 4096×4096, respectively). The errors for the CGNR and the BiCGStab2 family of iterations are plotted for all problem sizes. Colour versions of the figures presented in this paper are available at the Web page `www.math.sfu.ca/~mrt/Torus/`. Some preliminary results about this problem have also been reported in [19].

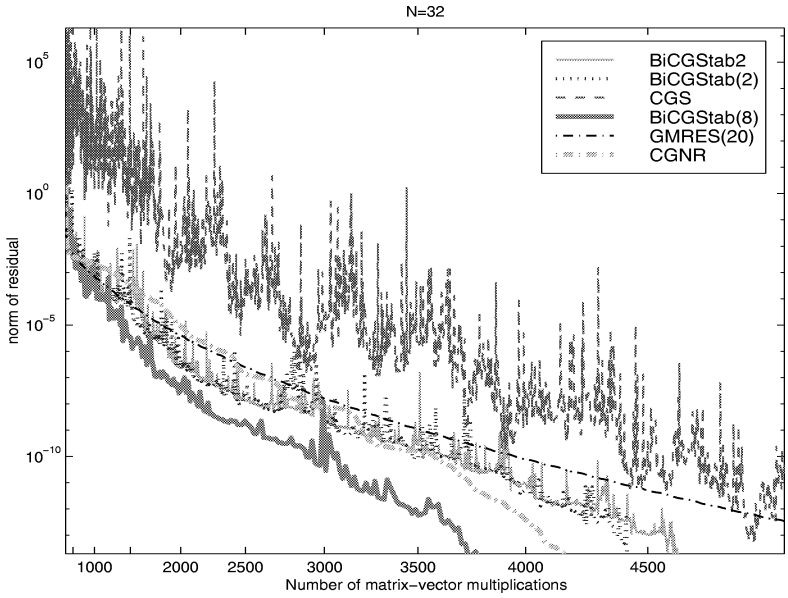

Fig. 9. Errors in BiCGStab2, BiCGStab(2), CGNR, CGS and GMRES(20) iteration, $N = 32$.

For $N = 32$ we show the results for BiCGStab2, BiCGStab(2), CGNR, CGS, and GMRES(20), as well as the best method, BiCGStab(8) in Fig. 9.

For $N = 16$ the CGS method is comparable to the BiCGStab2 family of methods; for $N = 32$ the residual is not reduced to below about $5 \cdot 10^{-14}$, as in the early stages of the iteration the residuals become as large as 10^8. This behaviour is even more pronounced for $N = 64$, when the residual gets as large as 10^{11} and cannot be reduced below 10^{-2}; even worse, in this case the residual climbs again to about 10^8. Thus we have a clear indication to avoid CGS (at least without preconditioning).

GMRES(20) exhibits the expected smooth monotone convergence behaviour, but proves not to be competitive with the other methods.

CGNR appears to need about two cycles of iterations (one cycle is N^2 iterations, which theoretically should give the exact solution) to get the error down close to machine precision. CGNR has monotone error behaviour, which is lost in BiCGStab2 and BiCGstab(ℓ). The work per step for each step in CGNR and BiCGStab2 is comparable, and approximately equal to two matrix–vector multiplies. The work for each step of BiCGstab(2) is about twice as much, and doubles again for BiCGstab(ℓ) for each doubling of ℓ. For $N = 16$ the CGNR method is fastest, but loses its speed advantage with increasing N. For BiCGstab(ℓ) the choice of ℓ does not appear to be significant for $N = 16$, however, for larger values of N, BiCGStab(ℓ) with $\ell = 8$ emerges as the best choice. The accuracy of BiCGStab(ℓ) deteriorates for $\ell = 16$, which may be related to the use of the modified Gram–Schmidt process in the BiCGStab(ℓ) algorithm. Beyond the value $\ell = 8$ the total flop count increases significantly. In all experiments we stopped the iteration when the norm of the *residual*, which can always be monitored, fell below 10^{-14}. It should be noted that these are results for "typical cases" capturing the qualitative differences between the methods, but the performance of the various methods does vary considerably depending on the specific system.

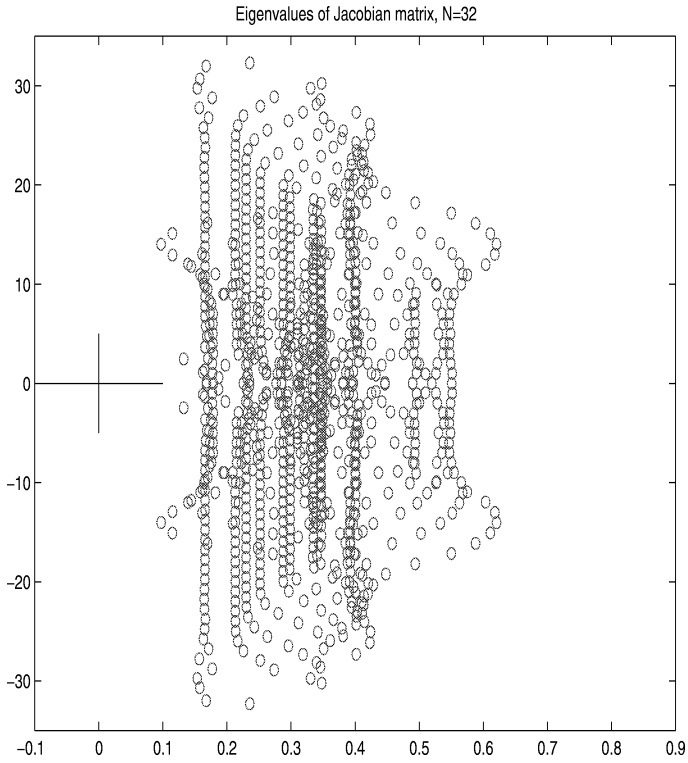

Fig. 10. Eigenvalues of a typical Jacobian, $N = 32$.

To shed some light on the convergence behaviour, we include a plot of the eigenvalues of a typical Jacobian matrix (for $N = 32$) in Fig. 10. The real parts of the eigenvalues are roughly between 0.1 and 0.62, whereas the imaginary parts do not significantly exceed ± 32i. Obviously, the matrix is "highly" unsymmetric. Its condition number is approximately 580, which is fairly small, and may explain the fairly good convergence behaviour of CGNR. Denoting by V the matrix of the normalized eigenvectors of J, we find that its condition number $\kappa(V)$ is roughly $4 \cdot 10^4$; this condition number measures the degree of nonnormality of J and features in many convergence estimates. More importantly, we have the somewhat unfortunate situation that the eigenvalues of the Jacobian do not exhibit any significant clustering behaviour; to the contrary, they are quite evenly distributed in the region of the spectrum, which is also perilously close to the imaginary axis. These properties make the iterative solution of the corresponding systems of equations very difficult.

As mentioned above one of the reasons for switching to iterative methods, is that from a certain size on the LU-factorization can no longer performed in core. Constant swapping of memory pages will basically grind the algorithm to a halt. Our iterative algorithms need $O(M)$ iterations, where $M = N^2$ is the order of the system of equations. Because of the sparse structure of our matrices, each matrix–vector product can be performed in roughly $4N^3$ floating point operations ($2N^3$ nonzero entries, each requiring 2 flops), as opposed to $2N^4$ flops for a full matrix. With $O(N^2)$ iterations, the total operation count for the iterative schemes is $O(N^5)$ as opposed to $O(N^6)$ for Gaussian elimination. There are two sensible ways to implement the matrix–vector products for our problems. One is to build and store the Jacobian

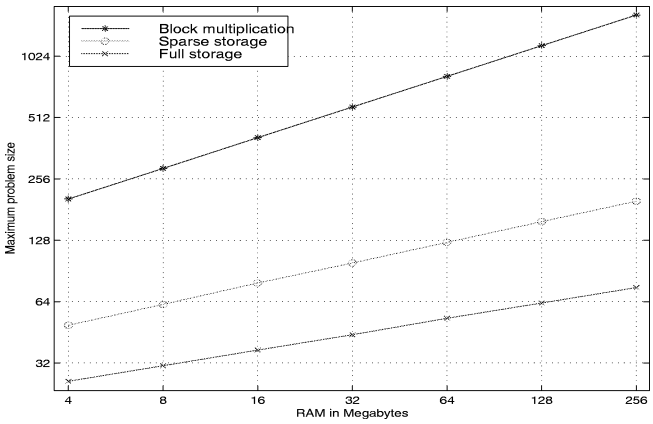

Fig. 11. Maximum problem size for in-core computation.

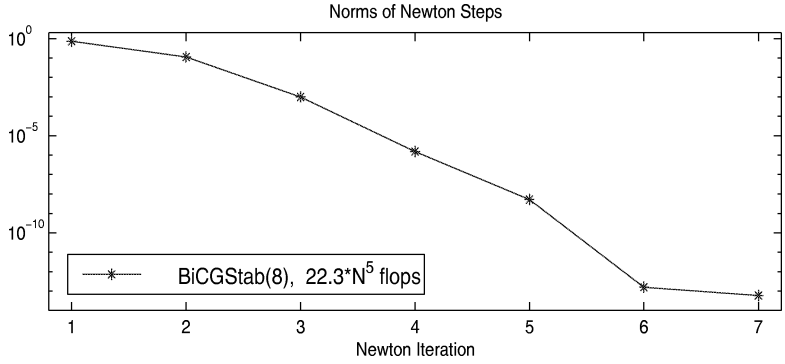

Fig. 12. Newton method with BiCGstab(8), $N = 32$.

matrix as a sparse matrix. This requires $2N^3$ storage locations, plus a similar amount of space for the index vectors. The other way is to only make use of the $N \times N$ matrix $\boldsymbol{\Lambda}$ of Eq. (3.2), and perform the multiplication with J for each of the terms in Eq. (3.4) separately. This has the advantage of reducing the storage requirement from $\mathrm{O}(N^3)$ to at most $\mathrm{O}(N^2)$, while only slightly increasing the cost of one matrix–vector from approximately $4N^3$ to roughly $4.5N^3$ flops. Storing the Jacobian as a sparse matrix, however, facilitates multiplication by its transpose which is needed in the CGNR algorithm. Fig. 11 gives a rough idea of the maximum values of N for a given amount of RAM for the various storage options which will not require out-of-core computations (i.e., memory page swapping).

Fig. 12 shows the results when we blend the outer Newton iteration for Eq. (3.3) with the inner iterative equations solver BiCGstab(8). The problem size is $N = 32$. Note that in the initial stages of the Newton iteration it is not necessary to compute highly accurate solutions to the linear system, however, the respective tolerance must be lowered as we move along in the Newton iteration. This can be achieved fairly well by using an error tolerance relative to the size of the right hand side of the system determining the Newton step. The solution we computed this way differs by less than 10^{-12} from the one computed with the direct matrix solver (see also Figs. 4 and 5). The total amount of work for this run was about

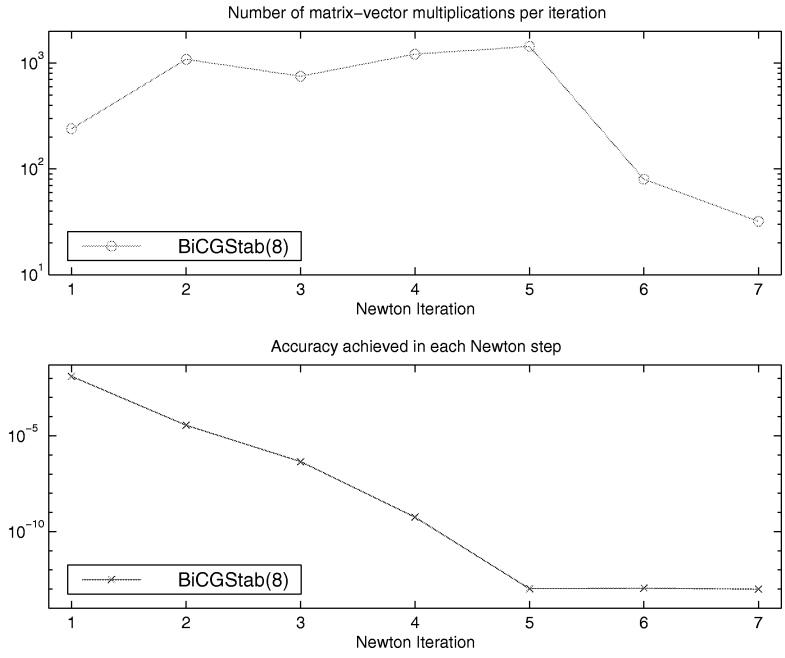

Fig. 13. Work per Newton step, and accuracy of BiCGstab(8), $N = 32$.

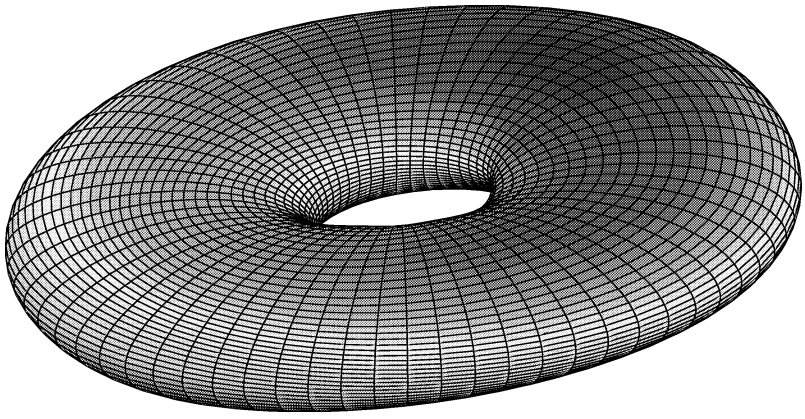

Fig. 14. Van der Pol oscillator, invariant torus, $N = 64$.

$22N^5$ flops compared to $3.1N^6$ for the sparse matrix solver, which amounts to a savings in flops by a factor of more than 4.5.

We should point out that the asymptotic operation counts for our iterative methods can be further improved by employing Fast Fourier Transforms to compute the convolution products with the matrix $\boldsymbol{\Lambda}$. These methods tend to become faster only for $N > 32$, so for the range of our values of N we do not expect to see significant savings.

6. Conclusions

We have demonstrated the viability and advantages of using spectral methods for the computation of invariant tori. For a theoretical convergence analysis see [14]. The large nonsymmetric systems arising in the Newton iteration although sparse, are ill-suited for direct solvers. They are best solved via iterative methods, as matrix–vector products are readily available and can be computed cheaply. The iterative method of choice for solving the linear system of equations arising in Newton's method is BiCGStab(ℓ), with our experiments clearly indicating $\ell = 8$ to give superior results. The BiCGStab2 family of methods appears to be very well suited to problems arising from first-order differential operators, as they seem to be able to accommodate spectra with large imaginary components. Preliminary results with preconditioners point to the possibility of a further reduction in the computational work by a factor of 10 or more.

Acknowledgements

I wish to thank Martin Gutknecht for suggestions and discussions on iterative methods for nonsymmetric systems, and Ronald Haynes for his help with producing Fig. 14.

References

[1] U.M. Ascher, R.M.M. Mattheij, R.D. Russell, Numerical Solution of Boundary Value Problems for Ordinary Differential Equations, Prentice-Hall, Englewood Cliffs, NJ, 1988.
[2] C. Canuto, M.Y. Hussaini, A. Quarteroni, T.A. Zang, Spectral Methods in Fluid Dynamics, Series of Computational Physics, Springer-Verlag, Heidelberg, 1988.
[3] L. Dieci, G. Bader, Solution of the Systems associated to Invariant Tori Approximation. II: Multigrid Methods, SIAM J. Sci. Comput. 15 (1994) 1375–1400.
[4] L. Dieci, G. Bader, Block iterations and compactification for periodic block dominant systems associated to invariant tori approximation, Appl. Numer. Math. 17 (1995) 251–274.
[5] L. Dieci, J. Lorenz, Block M-matrices and computation of invariant tori, SIAM J. Sci. Statist. Comput. 13 (1992) 885–903.
[6] L. Dieci, J. Lorenz, R.D. Russell, Numerical calculation of invariant tori, SIAM J. Sci. Statist. Comput. 12 (1991) 607–647.
[7] J.E. Dennis, R.D. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice-Hall, Englewood Cliffs, NJ, 1983.
[8] W.S. Don, A. Solomonoff, Accuracy and speed in computing the Chebyshev collocation derivative, SIAM J. Sci. Comput. 6 (1994) 1253–1268.
[9] R. Fletcher, Conjugate gradient methods for indefinite systems, in: G.A. Watson (Ed.), Numerical Analysis, Lecture Notes in Mathematics, Vol. 506, Springer, Berlin, 1976, pp. 73–89.
[10] J. Guckenheimer, P. Holmes, Nonlinear Oscillations, Dynamical Systems, and Bifurcations in Vector Fields, Applied Mathematical Sciences, Vol. 42, Springer-Verlag, New York, 1983.
[11] M.H. Gutknecht, Variants of BiCGStab for matrices with complex spectrum, SIAM J. Sci. Statist. Comput. 14 (1993) 1020–1033.
[12] R.D. Haynes, Invariant manifolds of dynamical systems: theory and computation, M.Sc. thesis, Simon Fraser University, Canada, 1998.

[13] C. Lanczos, Solution of systems of linear equations by minimized iterations, J. Res. Nat. Bureau Standards 49 (1952) 33–53.
[14] H. Mingyou, T. Küpper, N. Masbaum, Computation of invariant tori by the Fourier method, SIAM J. Sci. Comput. 18 (1997) 918–942.
[15] N.M. Nachtigal, S.C. Reddy, L.N. Trefethen, How fast are nonsymmetric matrix iterations?, SIAM J. Matrix Anal. Appl. 13 (1992) 778–795.
[16] G.L.G. Sleijpen, D.R. Fokkema, BiCGStab(ℓ) for linear equations involving unsymmetric matrices with complex spectrum, Electronic Trans. Numer. Anal. 1 (1993) 11–32.
[17] P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric linear systems, SIAM J. Sci. Statist. Comput. 10 (1989) 36–52.
[18] L.N. Trefethen, Spectra and Pseudospectra: The Behavior of Non-normal Matrices and Operators, book in preparation.
[19] M.R. Trummer, Unsymmetric systems arising in the computation of invariant tori, in: Copper Mountain Conference on Iterative Methods, 1994.
[20] H.A. van der Vorst, Bi-CGSTAB: A fast and smoothly convergent variant of Bi-CG for the solution of nonsymmetric linear systems, SIAM J. Sci. Statist. Comput. 13 (1992) 631–644.
[21] M. van Veldhuizen, A new algorithm for the numerical approximation of an invariant curve, SIAM J. Sci. Statist. Comput. 8 (1987) 951–962.