Using Python to Solve Partial Differential Equations

This article describes two Python modules for solving partial differential equations (PDEs): PyCC is designed as a Matlab-like environment for writing algorithms for solving PDEs, and SyFi creates matrices based on symbolic mathematics, code generation, and the finite element method.

Our work at the Simula Research Laboratory mostly focuses on computational applications in life sciences. Usually, this involves fairly typical partial differential equations such as the incompressible Navier-Stokes equations, elasticity equations, and parabolic and elliptic PDEs, but these PDEs are typically coupled either with each other or with ordinary differential equations (ODEs). Hence, even though the PDEs themselves are reasonably well understood, the couplings between them make the problems we study quite challenging.

Our design goals are therefore threefold. First, we want to easily define systems of PDEs. Second, we want it to be easy to play with different solution algorithms for systems of coupled PDEs. Finally, we want to reuse existing software to avoid reinventing the wheel.

We use many good and mature libraries from the Web, including Dolfin (www.fenics.org/dolfin/), GiNaC (www.ginac.de), MayaVi (http://mayavi.sourceforge.net), NumPy (http://numpy.scipy.org), PETSc (www.mcs.anl.gov/petsc/), SciPy (www.scipy.org), Trilinos (http://software.sandia.gov/trilinos/), and VTK (www.vtk.org). In fact, we're mixing these libraries with our own packages:

- Famms (verification based on the method of manufactured solutions),
- Instant (www.fenics.org/instant; inlining of C++ in Python),
- PyCC (http://folk.uio.no/skavhaug/heart_simulations.html; the underlying framework for gluing components together),
- PySE (http://pyfdm.sf.net; a finite difference toolbox),
- Swiginac (http://swiginac.berlios.de/; a Python interface to the symbolic mathematics engine GiNaC), and
- SyFi (www.fenics.org/syfi/; a finite element toolbox).

Some of these packages are Python modules, whereas the others—thanks to Python's popularity in scientific computing—are equipped with Python interfaces. By using Python, we don't have to mix these packages at the C level, which is a huge advantage.

1521-9615/07/$25.00 © 2007 IEEE
Copublished by the IEEE CS and the AIP

Kent-Andre Mardal, Ola Skavhaug, Glenn T. Lines, Gunnar A. Staff, and Åsmund Ødegård
Simula Research Laboratory

Solving Systems of PDEs

Currently, our most important application is in cardiac electrophysiology.[1] The central model here is the *bidomain model*,[2] which is a system of two PDEs with the following form:

$$\frac{\partial v}{\partial t} = \nabla \cdot (M_i \nabla v) + \nabla \cdot (M_i \nabla u) - I_{ion}(v, s), \qquad (1)$$

$$0 = \nabla \cdot (M_i \nabla v) + \nabla \cdot ((M_i + M_e)\nabla u). \qquad (2)$$

(The domain here is the same for both PDEs—that is, the heart—but it has two potentials, intra- and extra-cellular, which live inside and outside heart cells.) The primary unknowns here are the transmembrane potential v and the extra cellular potential u. The function $I_{ion}(v, s)$ describes the flow of ions across the cell membrane and can be quite complicated; M_i represents intracellular conductivity, and M_e is extracellular. The second argument $I_{ion}(v, s)$ is generally a vector of variables, governed by a set of ODEs:

$$\frac{\partial s}{\partial t} = F(s, t). \qquad (3)$$

Note that $s = s(x)$, so the ODE system is defined in each point (you can find examples of cell models at www.cellml.org). Hence, we must solve the system for each computational node.

The bidomain formulation gives an accurate description of the myocardial tissue's electrical conduction. Coupled with realistic ODE models of the ionic current, we can study a large range of phenomena, including conduction abnormalities due to ischmeia or channel myopathies (genetic defects), fibrillation and defibrillation, and drug intervention. Figure 1, for example, shows a geometric model of the human heart's ventricles. The color represents the transmembrane potential's magnitude; Figure 1a shows normal activation, and Figure 1b shows chaotic behavior (which corresponds to a fibrillatory heart with critically reduced pumping ability).

We solve the bidomain model in Equations 1 through 3 by using an operator-splitting approach, in which we first solve the ODE systems in each computational node at each time step before we solve the PDE system. Here's a simple Python script we use for solving this problem:

```
from dolfin import Mesh
from pycc.MatSparse import *
import numpy
from pycc import MatFac
from pycc import ConjGrad
from pycc.BlockMatrix import *
from pycc.Functions import *
from pycc.ODESystem import *
from pycc.CondGen import *
from pycc.IonicODEs import *
```

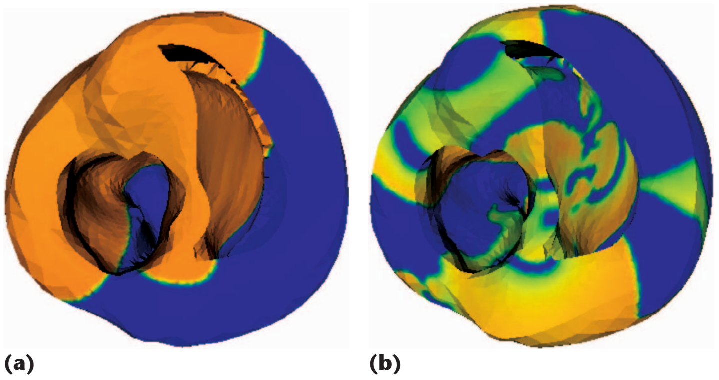


Figure 1. Human heart ventricles. This model compares (a) normal activity and (b) chaotic behavior.

```
mesh = Mesh("Heart.xml.gz")
matfac = MatFac.MatrixFactory(mesh)

M = matfac.computeMassMatrix()

pc = PyCond("Heart.axis")
pc.setconductances(3.0e-3, 3e-4)
ct = ConductivityTensorFunction(
   pc.conductivity)
Ai = matfac.computeStiffnessMatrix(ct)

pc.setconductances(5.0e-3, 1.6e-3)
ct = ConductivityTensorFunction(
   pc.conductivity)
Aie = matfac.computeStiffnessMatrix(ct)

# Construct compound matrices
dt = 0.1
A  = M + dt*Ai
B  = dt*Ai
Bt = dt*Ai
C  = dt*Aie

# Create the Block system
AA = BlockMatrix((A,B),(Bt,C))
prec = DiagBlockMatrix((MLPrec(A),
       MLPrec(C)))

v = numpy.zeros(A.n, dtype='d') - 45.0
u = numpy.zeros(A.n, dtype='d')
x = BlockVector(v,u)

# Create one ODE systems for each vertex
odesys = Courtemanche_ODESystem()
ode_solver = RKF32(odesys)
ionic = IonicODEs(A.n, ode_solver,
      odesys)
```

```
ionic.setState(odesys.getDefault
      InitialCondition())

# Solve
z = numpy.zeros((A.n,), dtype='d')
for i in xrange(0, 10):
    t = i*dt
    ionic.forward(x[0], t, dt)
    ConjGrad.precondconjgrad(prec, AA,
       x, BlockVector(M*x[0], z))
```

Although the code seems clean and simple, it's due to a powerful combination of C/C++/Fortran and Python. The script runs on desktop computers with meshes that have millions of nodes and can solve complete problems within minutes or hours. All computer-intensive calculations such as computing matrices, solving linear systems (via algebraic multigrid and the conjugate gradient method), and solving ODE systems are done efficiently in C or C++.

Creating Matrices for Systems of PDEs

We created the tool SyFi to define finite elements and variational forms, as well as generate C++ code for finite element computations. It uses the symbolic mathematics engine GiNaC and its Python interface Swiginac for all its basic mathematical operations. SyFi enables polynomial differentiation and integration on polygonal domains. Furthermore, it uses the computed expressions, such as entries in an element matrix, to generate C++ code.

The following example demonstrates how to compute an element matrix for the Jacobian of an incompressible power-law fluid's (nonlinear) stationary Navier-Stokes equations. Let

$$\mathbf{F}_i = \int_T (\mathbf{u}\cdot\nabla\mathbf{u})\cdot\mathbf{N}_i + \mu(\mathbf{u})\nabla\mathbf{u} : \nabla\mathbf{N}_i dx,$$

where

$\mathbf{u} = \sum_k u_k \mathbf{N}_k$ and $\mu(\mathbf{u}) = ||\nabla\mathbf{u}||^{2n}$.

Then,

$$\mathbf{J}_{ij} = \frac{\partial \mathbf{F}_i}{\partial u_j} = \frac{\partial}{\partial u_j}\int_T (\mathbf{u}\cdot\nabla\mathbf{u})\cdot\mathbf{N}_i + \mu(\mathbf{u})\nabla\mathbf{u} : \nabla\mathbf{N}_i dx, \quad (4)$$

Here's the corresponding Python code for computing Equation 4 and generating the C code:

```
from swiginac import *
from SyFi import *

def sum(u_char,fe):
    ujs = symbolic_matrix(1,fe.nbf(),
       u_char)
    u = 0
    for j in range(0,fe.nbf()):
        u += ujs.op(j)*fe.N(j)
    u = u.evalm()
    return u, ujs

nsd = cvar.nsd = 3
polygon = ReferenceTetrahedron()
fe = VectorCrouzeixRaviart(polygon,1)
fe.set_size(nsd) # size of vector
fe.compute_basis_functions()

# create sum u_i N_i
u, ujs = sum("u", fe)
n = symbol("n")
mu = pow(inner(grad(u), grad(u)),n)

for i in range(0,fe.nbf()):
  # nonlinear power-law diffusion term
  fi_diffusion = mu*inner(grad(u),
     grad(fe.N(i)))

  # nonlinear convection term
  uxgradu = (u.transpose()
     *grad(u)).evalm()
  fi_convection = inner(uxgradu,
     fe.N(i), True)

  fi = fi_diffusion + fi_convection

  Fi = polygon.integrate(fi)

  for j in range(0,fe.nbf()):
    # differentiate to get the Jacobian
    uj = ujs.op(j)
    Jij = diff(Fi, uj)
    print "J[%d,%d]=%s\n"%(i,j,
       Jij.printc())
```

Note that both the differentiation and integration is performed symbolically exactly as we would have done by hand. This naturally leads to quite efficient code compared to the traditional way of implementing such integrals—namely, as quadrature loops that involve the evaluation of basis functions, their derivatives, and so on. The `printc` function generates C++ code for the expressions; so far, we've used this system to generate roughly 60,000 lines of C++ code for computing various matrices based on various finite elements and variational forms.

To ease the integration of the generated C++ code in Python, we developed an inlining tool called

Instant, which lets us generate code, generate the corresponding wrapper code, compile and link it to an extension module, and then import the module on the fly. The following code demonstrates Instant with a simple example, in which we compute

$$y(x) = \frac{\partial}{\partial x}\sin(\cos(x))$$

symbolically, generate the corresponding C++ code, and inline the expression in Python with x as a NumPy array:

```
import swiginac as S
from Instant import inline_with_numpy
import numpy as N

x = S.symbol("x")
xi = S.symbol("x[i]")
f = S.sin(S.cos(x))

dfdx = S.diff(f,x)
print dfdx

string = """
void func (int n, double* x, int m,
double* y) {
  if ( n != m ) {
      printf("Both arrays should be of
         the same size!");
     return;
  }

  for (int i=0; i<n; i++) {
      y[i] = %s;
  }
} """ % dfdx.subs( x == xi ).printc()

print string

func = inline_with_numpy(string, arrays
      = [['n', 'x'], ['m', 'y']])

x = N.arange(100.0 )
y = N.zeros(100, dtype='d')

func(x,y)

print x
print y
```

We've shown that it's possible to solve real-life problems in a user-friendly environment by combining Python's high-level syntax with the efficiency of compiled languages. However, this approach opens up many new possibilities for combining symbolic mathematics and code generation, which is a largely overlooked alternative to traditional approaches in finite element simulations. We're currently implementing fairly advanced finite element methods such as the mixed elasticity method,[3] and we also want to simulate human tissue and blood with the most realistic models available today.

Acknowledgments

Mardal is supported by the Research Council of Norway under the grant ES254277.

References

1. J. Sundnes et al., *Computing the Electrical Activity in the Human Heart*, Monographs in Computations Science and Engineering, Springer-Verlag, 2006.
2. C.S. Henriquez, "Simulating the Electrical Behavior of Cardiac Tissue Using the Bidomain Model," *Crit. Rev. Biomedical Eng.*, vol. 21, no. 1, 1993, pp. 1–77.
3. D.N. Arnold, R.S. Falk, and R. Winther, "Finite Element Exterior Calculus, Homological Techniques, and Applications," *Acta Numerica*, 2006, pp. 1–155.

Kent-Andre Mardal *is a postdoc at the Simula Research Laboratory. His research interests include high-level numerical programming and solution of PDEs. Mardal has a PhD in scientific computing from the University of Oslo. Contact him at ken-and@simula.no.*

Ola Skavhaug *is a research scientist at the Simula Research Laboratory. His research interests include high-level numerical programming, PDEs, and code verification. Skavhaug has a PhD in scientific computing from the University of Oslo. Contact him at skavhaug@simula.no.*

Glenn T. Lines *is a research scientist at the Simula Research Laboratory. His research interests include PDEs and computer simulation of cardiac electrophysiology. Lines has a PhD in scientific computing from the University of Oslo. Contact him at glennli@simula.no.*

Gunnar A. Staff *is a consultant at Scandpower Petroleum Technology. His research interests include software for scientific computing and initial value problems. Staff has a PhD in scientific computing from the University of Oslo. Contact him at gst@scandpowerpt.com.*

Åsmund Ødegård *is an IT manager and part-time research scientist at the Simula Research Laboratory. His research interests include high-level languages in scientific computing, object-oriented numerics, PDEs, and high-level parallelism. Ødegård has a PhD in computational science from the University of Oslo. Contact him at aasmund@simula.no.*

Analysis of Functional Magnetic Resonance Imaging in Python

The authors describe a package for analyzing magnetic resonance imaging (MRI) and functional MRI (fMRI) data, which is part of the Neuroimaging in Python (NIPY) project. An international group of leading statisticians, physicists, programmers, and neuroimaging methodologists are developing NIPY for wider use.

Magnetic resonance imaging (MRI) measures induced magnetic properties of tissue. It has long been the chosen technique for creating high-resolution anatomical images of the human brain. Over the past decade, a new technique called *functional MRI* (fMRI) has become a powerful and widely used method for studying human brain function. fMRI measures regional blood flow changes in the brain, which can help researchers identify the most active brain areas during mental tasks such as memory and language.

Functional MRI Analysis

The first step of an fMRI analysis—image reconstruction—takes raw data from the scanner and performs a highly customized inverse Fourier transform to create a time series of 3D functional images. A typical next step is to estimate the movement between scans via an automated image-matching algorithm and then use that estimate to remove artifacts due to motion. Researchers commonly relate the activity detected in the low-resolution functional images to a high-resolution structural image of the same subject. However, to compare between subjects, the functional or structural data must be warped to match some standard brain, a process that requires sophisticated models of brain anatomy. Finally, statistical techniques are used to determine which brain regions are related to certain tasks or activities.

Clearly, fMRI data analysis comes with several challenges. First, it has a wide variety of computationally intensive spatial and statistical processing steps. Thus, an analysis software package must cover the range from file system and network interaction through complex image processing to advanced statistical inference and 3D visualization. Second, such analysis involves a massive volume of data, often reaching several hundred gigabytes.

The most common software package in use today is SPM (www.fil.ion.ucl.ac.uk/spm/), which is written in Matlab. Other common packages include FSL (www.fmrib.ox.ac.uk/fsl/) and AFNI (http://afni.nimh.nih.gov/afni/), written in C and C++, respectively. Although these packages are well-designed and supported, an increasing number of imaging scientists have found that Matlab is not powerful enough to support the industrial level of code size and complexity that neuroimaging requires, and that C and C++ are too low-level for rapid development.

1521-9615/07/$25.00 © 2007 IEEE
Copublished by the IEEE CS and the AIP

K. Jarrod Millman
University of California, Berkeley
Matthew Brett
MRC Cognitive Brain Science Unit, Cambridge, UK

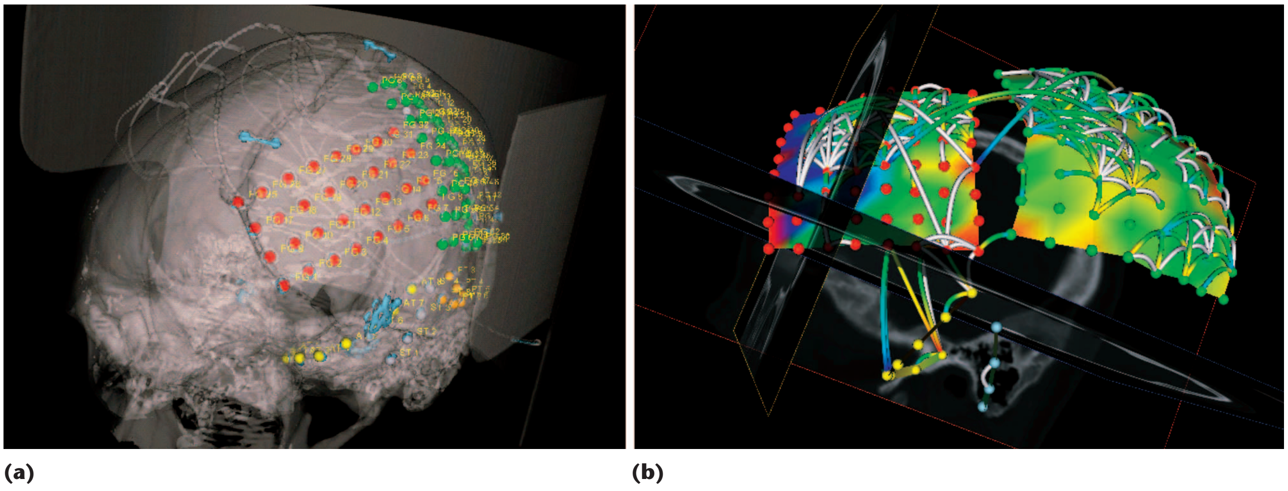


Figure 1. Python visualization widgets. Images from PBrain showing, (a) the position of the electrode arrays on the surface of the brain superimposed with a surface reconstruction of the skull from a CT scan and (b) measures of frequency and coherence of electrical activity overlaid on an image of the electrode positions.

Neuroimaging and Python

Python has become a natural choice for neuroimaging because it is high level, object-oriented, and interactive. All these features make it particularly well suited to scientific programming. It also has robust libraries for system interaction, image processing, matrix algebra, and visualization. Moreover, Python has very good tools for providing scripting support to software written in other languages. Finally, because of Python's strong integration with C and C++, it is often used as a type of high-level language glue for calling routines in a huge array of high-quality C/C++ libraries.

Accordingly, several significant neuroimaging packages have already been written in Python. Led by Daniel Sheltraw at Berkeley, researchers recently developed a set of Python tools for MRI reconstruction (https://cirl.berkeley.edu/view/BIC/ReconTools). John Hunter, the author of the matplotlib Python plotting library, developed PBrain, a sophisticated application for analyzing and visualizing the data from electrical recordings on the brain surface of epileptic patients (see Figure 1 and p. 90).[1] Finally, BrainVISA is a comprehensive pipeline-analysis tool for anatomical data, developed by a team of researchers in France (www.brainvisa.info).

Integrating Python Development in Neuroimaging

In this article, our main focus is on NIPY (http://neuroimaging.scipy.org), a new initiative to create a unified, open source, and open development environment for the analysis of neuroimaging data.[2] In particular, we focus on the fMRI component of NIPY, which is based on the BrainSTAT package that Jonathan Taylor wrote at Stanford University.[3] We are also working with Hunter and researchers at the University of Chicago to better integrate PBrain into the NIPY framework.

Image Model

An MRI scanner produces images that represent slices of brain tissue, and several of these slices together constitute a 3D whole-brain scan. The values in each voxel (volume element) in a 3D image slice represent measurements from a small volume of the brain.

A major problem in neuroimaging is that different analysis packages and scanners use different output image formats that are not readily compatible. To address this, NIPY provides read and write capabilities for all the popular image formats in neuroimaging, as well as access to binary data images and Python arrays in memory. Images can be compressed upon read or write, loaded from an arbitrary URL (with local caching), and managed by memory mapping where possible. Images also have iterators for reading data in slices and other subsets; Python iterators offer an elegant mechanism for constructing lazily instantiated sequential data structures—a perfect abstraction for intuitively representing sequential data (such as time series, spatial slices, or generally $(n - 1) - d$ data bricks from an $n - d$ data set) without sacrificing memory efficiency.

Spatial transforms on images are fundamental to neuroimaging—for movement correction, warping to templates, and many other analysis steps. NIPY

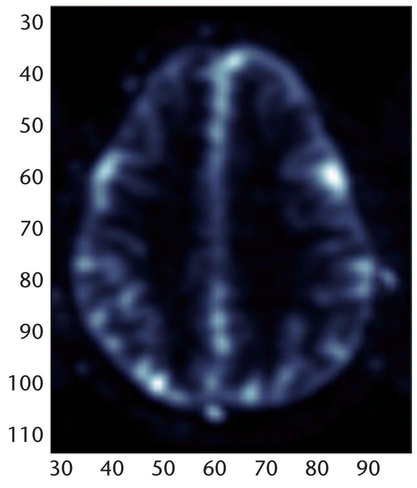


Figure 2. Image of standard deviation across time points. By summarizing data, the standard deviation image can highlight problems of background noise and artifacts that vary over time.

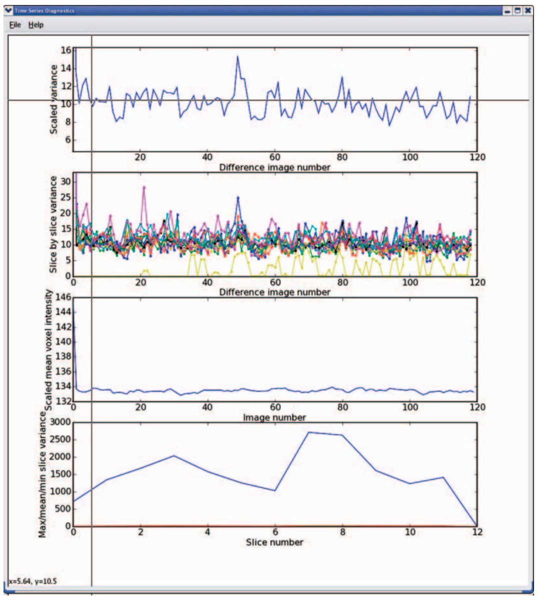


Figure 3. Time-series diagnostics. Researchers can use these four plots to diagnose potential problems in the time-series data.

offers a general model of image spatial transforms that allow arbitrary combinations of linear and nonlinear transforms, volume-to-surface mappings, and flexible levels of image interpolation. A common analysis technique is to examine the signal from a restricted region of the brain, or *region of interest* (ROI). NIPY has several ROI objects and functions, including discrete (point-level) and continuous (sphere, ellipse) region definitions, region combination algebra, and region data extraction.

Data Diagnostics

Because the scanner's signal quality can vary from day to day, special attention is needed to ensure good data. Currently, NIPY offers several diagnostic tools to let researchers discover problems with their data before they analyze it.

Images that summarize data across a time series provide a straightforward way to examine fMRI data. Taking an image of the standard deviation of signal intensity across time, for example, can highlight problems in the time series acquisition that occur with subject movement or instabilities of the data acquisition over time (see Figure 2).

Figure 3 shows four plots that can help diagnose potential problems in the time series. The top plot displays the scaled variance from image to image; the second plot shows the scaled variance per slice; the third plot shows the scaled mean voxel intensity for each image; and the bottom one plots the maximum, mean, and minimum scaled slice variances per slice.

Another powerful technique for data diagnosis is principal components analysis (PCA), which is particularly useful for detecting outlying time points or unexpected sources of spatially coherent noise. PCA on an image time series takes the time points as rows and voxels as columns, and decomposes the data into components that can efficiently express the source of variance in the data. Each component consists of a characteristic time series and the voxel weights contributing to that component, and these weights are viewable as an image. Figure 4 shows images of the weights for the first four principal components of a functional data set.

Statistical Processing

Subjects in a typical fMRI experiment perform some task or receive stimuli while being scanned; thus, areas activated by the task or stimulus exhibit voxel time series correlated with the experimental design. Unfortunately, noise in fMRI analysis can come from multiple sources, and the signal is relatively low. This problem with signal detection has led to several standard and more complex statistical methods.[4] Partly for historical reasons, current analysis packages use nonstandard or low-level statistical terminology and interfaces, making them less accessible to scientists with general training in statistics.

At Stanford, Taylor developed a general set of statistical model objects in Python that use standard statistical terminology and allow flexible high-level statistical designs. Similar to the R statistical language, these objects implement a general series

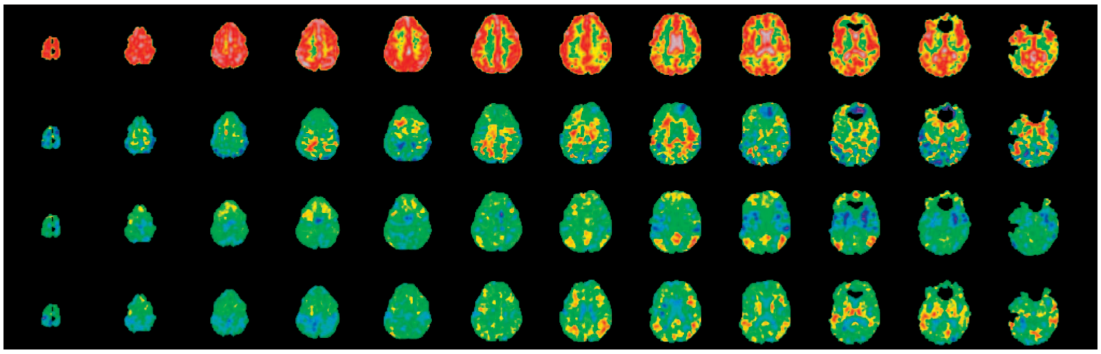


Figure 4. Principal components analysis (PCA). This type of analysis is particularly useful for detecting outlying time points or unexpected sources of spatially coherent noise.

of statistical models and contrasts for data, including functional images. These models form the basis of NIPY statistical analysis and are now maintained as part of the SciPy distribution; we hope these will attract further development from researchers outside brain imaging.

Python's power and generality mean that we have many fruitful directions in which to take NIPY. At the most basic, its high-level language features make it much easier to refactor the code into a well-patterned, high-level interface that scientific developers can quickly pick up. Because Python has such good integration with C/C++, we also have ready access to very powerful visualization and image-processing libraries. Two very important examples are the Visualization Toolkit (VTK) and the Insight Toolkit (ITK). VTK provides high-quality 3D graphical display and includes Python wrappers as part of its standard distribution, whereas ITK is a library of image registration and segmentation routines originally developed for the Visible Human Project. Like VTK, Python wrappers are part of the standard ITK distribution.

We intend NIPY to become the standard analysis library in neuroimaging in the medium term, which means we will need to provide the ability to call routines in other packages that are more familiar to researchers. Thankfully, this is a much easier task in Python than in many other languages because of its ability to interact with languages such as C.

Python's language features can also help us tackle the problem of provenance tracking. Because imaging analyses and data sets are very diverse, researchers use a variety of analysis packages and rarely record all their data and analysis parameters, making it very difficult for other people to reproduce published analyses. Fortunately, Python has excellent support for metaprogramming techniques (including metaclasses and function decorators) that can transparently change object behavior. We can thus use these techniques with Python's object introspection to capture the data's nature and processing in a way that can be closely wedded to the analysis.[5] Ultimately, this means that the analysis can become self-documenting, making it far easier to reproduce.

References

1. J.D. Hunter et al., "Locating chronically implanted subdural electrodes using surface reconstruction," *Clinical Neurophysiology*, vol. 116, no.8, 2005, pp. 1984–7.
2. J.E. Taylor et al., "BrainPy: An Open Source Environment for the Analysis and Visualization of Human Brain Data," *Neuroimage*, vol. 26, no. 763, 2005, pp. T–AM.
3. J.E. Taylor and K.J. Worsley, "Inference for Magnitudes and Delays of Responses in the FIAC Data Using BRAINSTAT/FMRISTAT," *Human Brain Mapping*, vol. 27, no. 5, 2006, pp. 434–441.
4. A.W. Toga and J.C. Mazziotta, eds., *Brain Mapping: The Methods*, 2nd ed., Elsevier Science, 2002.
5. K.J. Millman and M. D'Esposito, "Data and Analysis Management for Functional Magnetic Resonance Imaging Studies," *Proc. Int'l Advanced Database Conf.*, M. Amin et al., eds., US Education Service, 2006, pp. 24–28.

***K. Jarrod Millman** is the director of computing for the Helen Wills Neuroscience Institute at the University of California, Berkeley. His research interests include functional brain imaging, informatics, configuration management, and computer security. Millman has a BA in mathematics and computer science from Cornell University. Contact him at millman@berkeley.edu.*

***Matthew Brett** is a senior investigator scientist at the Medical Research Council (MRC) Cognition and Brain Science Unit in Cambridge, UK. His research interests include functional brain imaging and localization of brain function. Brett has an MD from Cambridge University. Contact him at matthew.brett@gmail.com.*

Python for Internet GIS Applications

Python offers a unique capability in the field of geographic information system applications because it helps developers create multipurpose Internet maps. This article discusses PDF maps in particular.

Researchers have used Python extensively in desktop geographic information system (GIS) applications for data processing, analysis, management, spatial statistical analysis, and so on since ESRI first released the ArcGIS suite, version 9.0.1, in January 2005. However, Python's full value and potential hasn't been fully explored or deployed in Internet-based GIS applications yet. This article describes how Python could offer a unique and easy approach for Internet GIS developers.

Interactive mapping is an important part of Internet-based GIS applications. Typically, Web users activate the print function in their browser's menu to print out a map on a particular Web page, but generating high-quality PDF map products has been a challenge for developers. Although the ArcMap Image Server in ESRI's ArcIMS has a function for printing PDF maps, it requires the ArcMap license, and the performance isn't very satisfactory because it takes so long to get the PDF product (see www.esri.com/software/arcgis/arcims/). Furthermore, this function doesn't support the fusion of ArcIMS image products with the Web Mapping Service's (WMS's) map images. Although PDF output support is part of the latest version of MapServer (an open source map software package for the Web; http://mapserver.gis.umn.edu/docs/howto/pdf-output), MapServer itself doesn't support WMS map images or the maps' surrounding components (the legend, scale, and so on) in its PDF map product.

Using some of the "batteries" included with the Python language, we can easily create PDF maps and reports, make composite images that integrate WMS map images with the map image output, and develop elevation profiles. Python's capability for multipurpose Internet mapping applications is unique yet easy to deploy.

1521-9615/07/$25.00 © 2007 IEEE
Copublished by the IEEE CS and the AIP

Xuan Shi
West Virginia University

Generating a Map Document with Python

Python uses the `Reportlab` and `Image` module libraries to make PDF map creation an easy process. Although `Reportlab` isn't included with Python, it's available as a free downloaded at www.reportlab.org/downloads.html and can be integrated in the server machine's Python directory. Application developers must first determine the location of the map image product generated by the Internet GIS server, but creating a PDF map document with Python is extremely simple:

```
img=Image.open(mapDir)
c = canvas.Canvas(pdfDir,
pagesize=landscape(A4))
c.drawInlineImage(img, 83, 80,
img.size)
c.showPage()
c.save()
```

In this code snippet, `mapDir = "C:\ArcIMS\output\WVBaseMap_14384368425964.png"`, whereas `pdfDir ="C:\ArcIMS\output\WVBaseMap_14384368425964.pdf"`.

We can use the same approach to insert a logo, map legend, scale bar, arrows, and so forth in the PDF document, as Figure 1 shows. We can also easily add multiple pages to the PDF document by calling the `showPage()` and `save()` functions multiple times. With the `Reportlab` toolkits, developers can add text, graphics, lines, and polygons into the PDF map document. Because Python creates PDF map documents by directly processing the map image output generated by the Internet GIS server, all application developers can deploy this approach, regardless of their server environment.

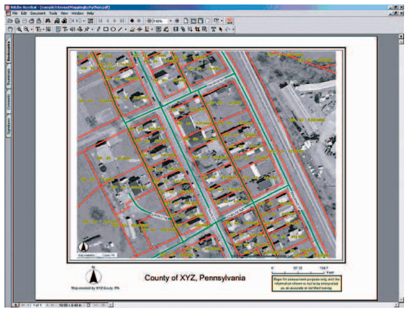


Figure 1. PDF map with varied surrounding components. Using Python, developers can add legends, scale bars, and arrows to maps.

Image Fusion by Python

The Open Geospatial Consortium (OGC; www.opengeospatial.org) developed the WMS specification to facilitate geospatial data interoperability and reusability. In this way, users can access map and geospatial data without having to save the data set locally. Using a specially structured HTTP request, they can retrieve map images and integrate them into their own applications as background images. But when a user wants to create a PDF map document, we have to create a composite image that merges multiple images.

Generally, we can overlay and merge multiple WMS-compatible map images as a single map image in a Web browser because such images are normally transparent. The composite raster image in Figure 2a, for example, looks like one map image, but it's actually composed of two images, as Figure 2b (a WMS map generated by Microsoft's TerraServer) and Figure 2c (a transparent PNG map image generated by ArcIMS) show. In this case, the WMS image is served as the background image in a Web page that contains the ArcIMS map image output. Because TerraServer's WMS image is retrieved remotely by an HTTP `GETMAP` request and served as a background image on the Web browser, to create a PDF document with composite images, we must first save the background image into the file directory with the map output generated by the local Internet mapping server.

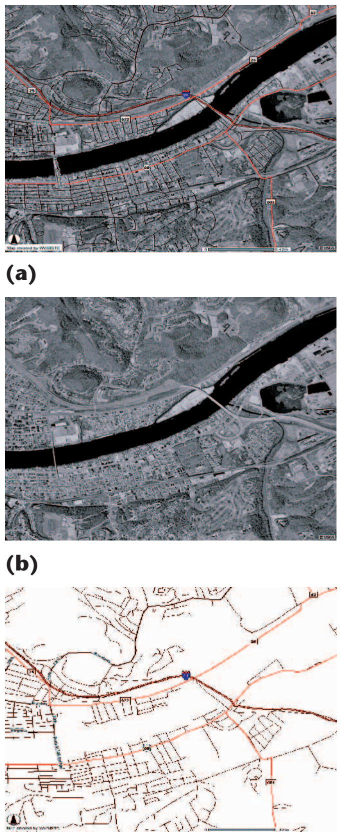


Figure 2. Composite image overlay. What looks like (a) a composite raster image in a Web browser is actually (b) a WMS map image generated by TerraServer and (c) a PNG map output generated by ArcIMS.

The WMS specification provides a basic guide

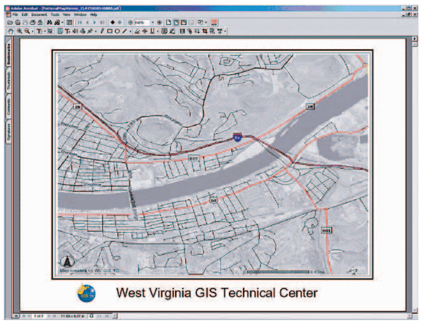


Figure 3. Composite image. The ArcIMS map is transparent with a white background.

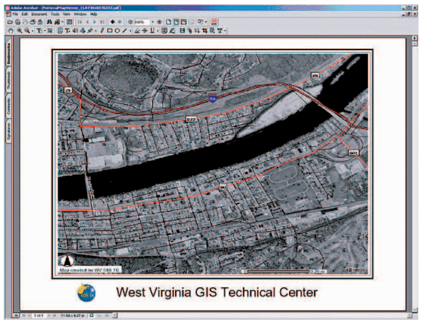


Figure 4. Composite image with transparent ArcIMS map. After we get this image, we can create a PDF map document.

for image registration that helps developers retrieve the correct WMS map images with the exact same scale and extent of the corresponding map output generated by the local server. Whenever the user browses the Web, ArcIMS will generate four variables that define the bounding box (`BBOX`) of the map extent—`MinX`, `MinY`, `MaxX`, and `MaxY`—as well as the map output image's width and height (`mWidth`, `mHeight`). These variables can then be used to dynamically retrieve the background TerraServer Digital Orthophoto Quarter (DOQ) image via an HTTP request:

```
http://terraserver-
usa.com/ogcmap.ashx?version=1.1.1&RE-
QUEST=GetMap&
BBOX=MinX,MinY,MaxX,MaxY&width=mWidth&
height=mHeight&LAYERS=DOQ&styles=&SRS=
EPSG:26917&format=PNG&transparent=true
```

When ArcIMS generates its map output image by default into server output directory `C:\ArcIMS\output\` with a specific code (such as `WVBaseMap_14384368425964.png`), we can also use such code to name the TerraServer image in the same file directory—for example, `terraDOQDir = "C:\ArcIMS\output\WVBaseMap_14384368425964terraDOQ.png"`. In this way, we understand that we can use these two images to create the composite map image for the PDF document.

Python has library modules that provide an image-fusion function to generate a PDF map with composite images. If the user has requested a WMS map service such as Microsoft's TerraService, we first need to use the `urllib` and `urllib2` modules in Python to retrieve the WMS image from the external server, and then save the TerraServer DOQ image to the local machine with the following code:

```
response = urllib2.urlopen(terraDO-
Qurl)
mapImg = response.read()
fo = open(terraDOQDir,'wb')
fo.write(mapImg)
fo.close()
```

To create a composite image that merges the ArcIMS transparent map image with the TerraServer background image, however, the Python function for image fusion has different behavior when the mask is defined differently in function `Image.composite(image1, image2, mask)`. By creating a composite image with the following Python script, the entire ArcIMS map image is transparent, with a white background in the composite image (see Figure 3):

```
w,h = image1.size
mask = Image.new("L",(w,h))
draw = ImageDraw.Draw(mask)
draw.rectangle([0, 0, w, h],
fill="#FF0000")
img =
Image.composite(image1,image2,mask)
```

We want to enforce that we can set the ArcIMS image's transparent part as a transparent window by using the following Python script for implementation in other Internet GIS applications. In this way, the white background is removed, and the original image's transparent behavior is maintained. After we get the composite image, we can create a PDF map document, as in Figure 4:

```
imgTerraDOQ=Image.open(terraDOQDir)
imgWVmap=Image.open(mapDir)
white =
Image.new("RGB",imgWVmap.size,(255,255
,255))
imgWVmap=imgWVmap.convert("RGBA")
r,g,b,a = imgWVmap.split()
img = Image.composite(imgWVmap, imgTer-
raDOQ, a)
```

Generating Elevation Profiles with Python

Elevation profiles show the change in elevation along a line. They can help people assess a trail's difficulty, for example, or evaluate the feasibility of placing a road along a given route. Figure 5 shows a map composed of a colorful digital elevation model's (DEM's) raster data layer. When the user clicks on the Internet map viewer, he or she can retrieve the elevation value at that (x, y) coordinate. To generate an elevation profile, as in Figure 6, we retrieve multiple elevation values along the line by dividing it into multiple sections and then using Python to create an elevation profile map as a PDF.

Figure 6 shows two elevation profiles within a single graphic. The red line shows the true elevation information between the start and end points that the user provided in Figure 5. Because the horizontal distance is roughly 21.17 miles, the vertical elevation information is exaggerated. The brown line shows the true landscape along the line segment because its value is recalculated dynamically based on the horizontal distance and the ratio used to adjust the vertical elevation.

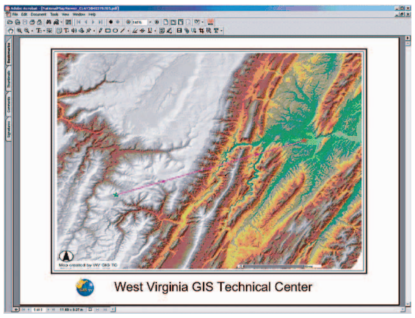


Figure 5. Elevation map. The digital elevation model's raster layer focuses on eastern West Virginia.

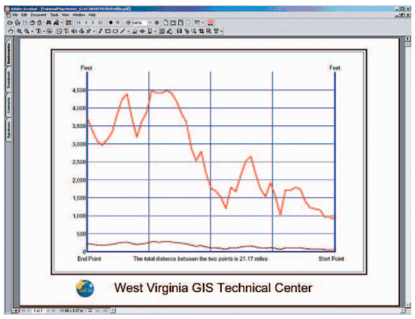


Figure 6. Elevation profile along the line shown in Figure 5. The red line shows the true elevation information, and the brown line shows the true landscape.

Python's large "batteries included" set of library modules can help us easily perform very sophisticated maneuvers with very little programming. Although this article uses ArcIMS as an example platform for Internet GIS application development, the same approach I discussed can be used with MapServer and other platforms. Because Python offers strong support for integration with other languages and tools, we can expect Internet GIS developers to continue expanding the use and scope of Python applications in the coming months and years.

Acknowledgments

I gratefully acknowledge Paul Dubois's kind review and advice for this article.

***Xuan Shi** is a PhD candidate in the Department of Geology and Geography at West Virginia University. His research focuses on implementing Web services technology in GIS applications. Contact him at Xuan.Shi@mail.wvu.edu.*

Quantum Chaos in Billiards

An important class of systems—billiards—can show a wide variety of dynamical behavior. Using tools developed in Python, researchers can interactively study the complexity of these dynamics. Such behavior is directly reflected in properties of the corresponding quantum systems, such as eigenvalue statistics or the structure of eigenfunctions.

Chaotic behavior in dynamical systems is a well-studied phenomenon. A particularly illustrative class of such systems is the so-called *billiard systems*, in which a point particle moves freely along straight lines inside a two-dimensional domain Ω with elastic reflections at the boundary. Billiards are interesting because they're mathematically well studied with various rigorous results, so they provide a good basis for investigating the implications of classical chaos in quantum mechanical systems. Apart from the basic research aspect, though, possible applications range from the design of semiconductor nanostructures to optical microlasers.

In this article, I give a brief overview of some aspects of our research at the Technische Universität Dresden, where numerical computations play an important role and Python allows for an efficient way of implementation.

Billiard dynamics

The boundary determines a billiard's dynamical properties. Figure 1 demonstrates this, showing 50 iterations of an initial point for two billiards, parametrized in polar coordinates by $\rho(\varphi) = 1 + \varepsilon \cos(\varphi)$ with $\varphi \in [0, 2\pi]$ for parameters $\varepsilon = 0$ (circular billiard) and $\varepsilon = 1$ (cardioid billiard).[1] The circular billiard is an example of an integrable system showing regular dynamics. The opposite extreme is the cardioid billiard, which is fully chaotic—this means that nearby trajectories separate exponentially as a function of time (hyperbolicity) and that a typical trajectory will uniformly fill out the available space (ergodicity).

Because the billiard's motion follows a straight line, it's convenient to use the boundary to define a Poincaré section,

$$\mathcal{P} := \{(s, p) \mid s \in [0, |\partial\Omega|], p \in [-1, 1]\}, \qquad (1)$$

where s is the arc length along the boundary $\partial\Omega$, and $p = \langle \boldsymbol{v}, \boldsymbol{T}(s)\rangle$ is the projection of the unit velocity vector $\boldsymbol{v}$ after the reflection onto the unit tangent vector $\boldsymbol{T}(s)$ in point $s \in \partial\Omega$. We get a Poincaré map P of a point $\xi = (s, p) \in \mathcal{P}$ by considering the ray starting at point $\boldsymbol{r}(s) \in \partial\Omega$ in the direction specified with p and then determining the first intersection with the boundary, which ultimately leads to a new point, $\xi' = (s', p')$. Explicitly, the velocity in the $\underline{\boldsymbol{T}, \boldsymbol{N}}$ coordinate system is given by $(p, n = \sqrt{1-p^2})$, so in Cartesian coordinates,

$$\begin{aligned} \boldsymbol{v} = (v_x, v_y) &= \begin{pmatrix} T_x & N_x \\ T_y & N_y \end{pmatrix} (p, n) \\ &= (T_x p + N_x \sqrt{1-p^2}, T_y p + N_y \sqrt{1-p^2}). \qquad (2) \end{aligned}$$

Numerically, the main task here is to find the next intersection for a given starting point on the boundary and direction, specified as s and p. If an implicit equation,

1521-9615/07/$25.00 © 2007 IEEE
Copublished by the IEEE CS and the AIP

ARND BÄCKER
Technische Universität Dresden

$$F(x, y) = 0, \tag{3}$$

determines the boundary, we can determine the new point $\boldsymbol{r}'$ by solving

$$F(x + tv_x, y + tv_y) = 0 \tag{4}$$

for $t > 0$. In the case of the circular billiard, we can easily obtain an analytical solution that will lead to an explicit prescription for the billiard mapping. In general, however, only a numerical solution to Equation 4 is possible.

For nonconvex billiards, there are points $\xi = (s, p) \in \mathcal{P}$ for which there is more than one solution of Equation 4 (apart from $t = 0$); obviously, we must choose the one with the smallest $t > 0$. We can sometimes use the condition in Equation 3 to remove the $t = 0$ solution analytically from Equation 4. If F is a polynomial in x and y, this reduces the order of Equation 4 by one—for example, for the cardioid billiard to get a cubic equation for t.[2] From that solution, we can get the coordinates $(x', y') = (x, y) + t\boldsymbol{v}$.

Numerically, though, we must find one or several solutions to Equation 4, depending on the type of boundary, so we should try to bracket the zero with the smallest t, such as by evaluating $F(x + tv_x, y + tv_y)$ for sufficiently many values of t, and then use `scipy.optimize.brentq` to determine the zero as precisely as possible. We must pay special attention to glancing motion—when p is near ± 1—because t can get very close to 0. Moreover, in nonconvex billiards, zeros of $F(x + tv_x, y + tv_y)$ can get very close to each other and are therefore easily missed.

Visualization with Python

Both for research and teaching, it's extremely useful to interactively explore the dynamics in billiards by using visualizations of the trajectories specified in the Poincaré section via the mouse. For this purpose, my students and I developed an application written in Python, using wxPython (www.wxpython.org) for the GUI and a special widget to quickly plot several points (www.physik.tudresden.de/~baecker/python/plot.html). Figure 2 shows a typical screenshot ($\varepsilon = 0.3$). In this case, the system shows both regular and irregular motion, depending on the initial point.

Quantum Billiards

Although classical mechanics can correctly describe macroscopic objects, a quantum mechanical description is necessary at small scales. Due to the Heisenberg uncertainty principle, it's impossible

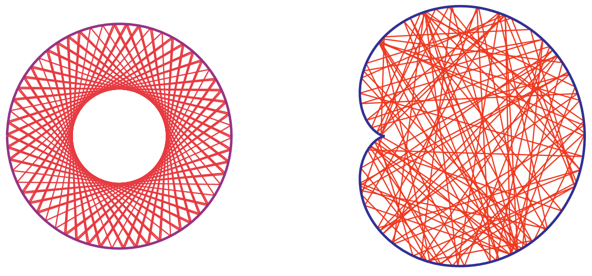


Figure 1. Billiard dynamics. Note the difference between (a) the circular billiard and (b) the chaotic dynamics in the cardioid billiard.

to simultaneously specify a particle's position and momentum—instead, the particle state is specified with a wavefunction whose absolute value squared is interpreted as the probability density. For quantum billiards, finding the stationary solutions of the Schrödinger equation reduces to the determination of eigenvalues and eigenfunctions of the Helmholtz equation,

$$-\Delta\psi_n(\boldsymbol{q}) = E_n\psi_n(\boldsymbol{q}), \quad \boldsymbol{q} \in \Omega, \tag{5}$$

with, for example, Dirichlet boundary conditions—that is, $\psi_n(\boldsymbol{q}) = 0$ for $\boldsymbol{q} \in \partial\Omega$. Here, Δ denotes the Laplace operator, which reads in two dimensions

$$\Delta = \left(\frac{\partial^2}{\partial q_1^2} + \frac{\partial^2}{\partial q_2^2}\right).$$

The interpretation of ψ is that $\int_D |\psi(\boldsymbol{q})|^2 d^2q$ is the probability of finding the particle inside the domain $D \subset \Omega$.

For some simple domains Ω, it's possible to solve Equation 5 analytically. For the billiard in a rectangle with sides a and b, for example, the (non-normalized) eigenfunctions are given by $\psi_{n_1,n_2}(\boldsymbol{q}) = \sin(\pi n_1 q_1/a)\sin(\pi n_2 q_2/b)$ with corresponding eigenvalues

$$E_{n_1,n_2} = \pi^2(n_1^2 / a^2 + n_2^2 / b^2)$$

and $(n_1, n_2) \in \mathbb{N}^2$. For the billiard in a circle, the eigenfunctions are given in polar coordinates by $\psi_{mn}(r, \varphi) = \mathcal{J}_m(j_{mn}r)\exp(im\varphi)$, where j_{mn} is the nth zero of the Bessel function $\mathcal{J}_m(x)$ and $m \in Z$, $n \in N$. Using `scipy.special.jnzeros`, we can easily obtain a given number of zeros of $\mathcal{J}_m(x)$ for fixed m.

In general, though, no analytical solutions of Equation 5 exist, so we must use numerical methods to compute eigenvalues and eigenfunctions. Among the many different possibilities, the so-

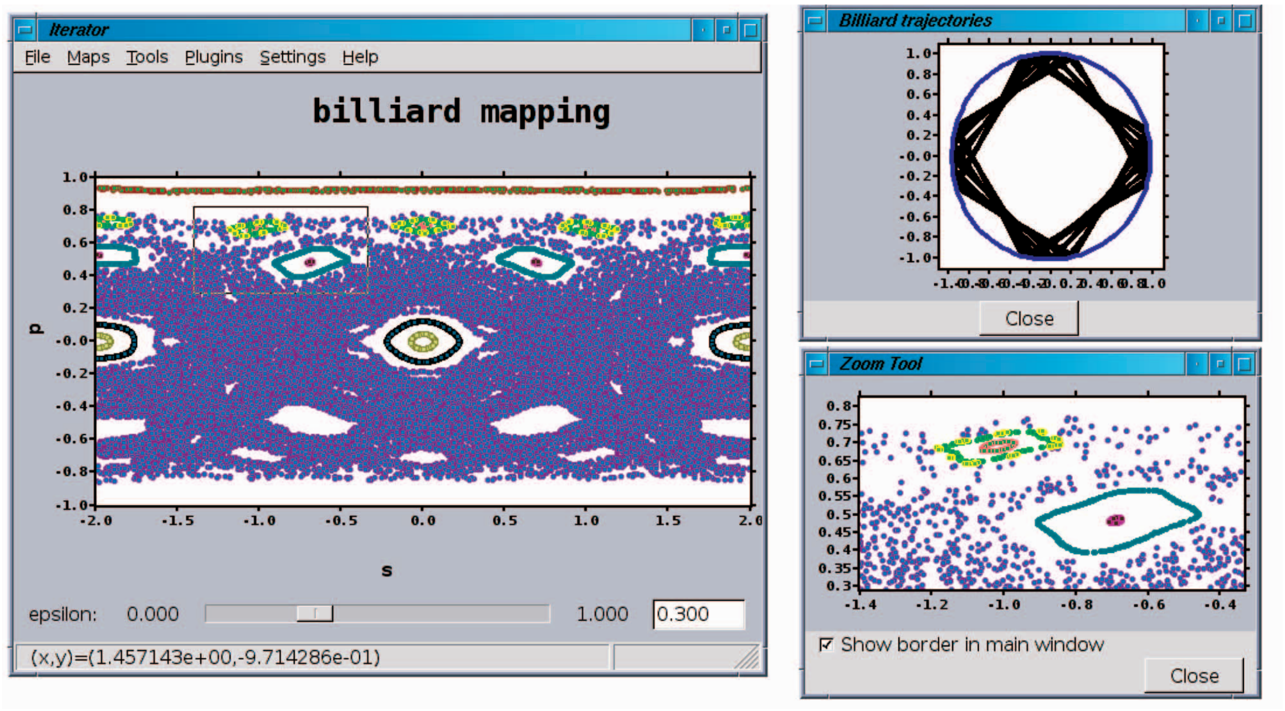

Iterator
File Maps Tools Plugins Settings Help
billiard mapping
p
s
epsilon: 0.000
1.000
0.300
(x,y)=(1.457143e+00,-9.714286e-01)
Billiard trajectories
Close
Zoom Tool
Show border in main window
Close

Figure 2. Screenshot of a Python application. To interactively explore billiard systems, we can define initial conditions in the Poincaré section on the left, for which the iterates are computed and the corresponding trajectory is shown in position space on the right. Magnified views of parts of the Poincaré section are possible, as shown in the lower right window.

called boundary-integral method is popular.[3] By using Green's theorem, we can transform the two-dimensional problem in Equation 5 into a one-dimensional integral equation. Discretization leads to a matrix equation for which we must determine the zeros of a determinant as a function of energy E. To detect nearly degenerate energy levels, it's most efficient[4] to use a singular value decomposition. For the numerical implementation, this is the most time-consuming step, which involves running `scipy.linalg.flapack.zgesdd`. Using this approach, we can easily compute 2,000 to 10,000 (and more, if necessary) eigenvalues and eigenfunctions. In particular, due to the computation's independence at different E, we can parallelize this problem rather straightforwardly by using as many CPUs as available. Communication between different CPUs isn't needed, only the initial value of E must be transferred to each CPU (via the message-passing interface, for example).

Quantum Chaos

A fundamental question in quantum chaos is the impact of the underlying classical dynamical properties on the statistical behavior of eigenvalues. It has been conjectured that the statistics of random matrices obeying appropriate symmetries can describe the eigenvalue statistics of fully chaotic systems.[5] For generic integrable systems, we expect a Poissonian random process to describe the energy-level statistics.[6]

The simplest spectral statistics is the level-spacing distribution $P(s)$ obtained from the histogram of the spacings

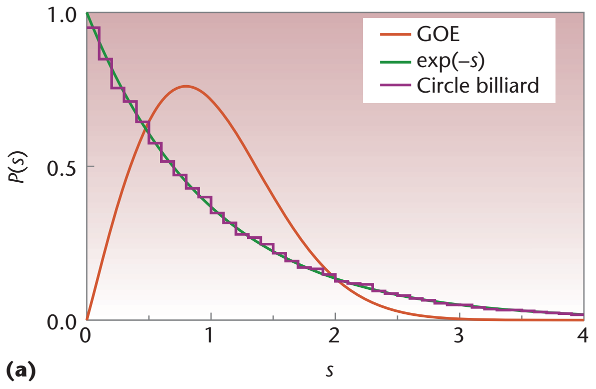
$$s_n := x_{n+1} - x_n\,, \tag{6}$$

where x_n are rescaled eigenvalues such that their average spacing is 1. Once we compute the eigenvalues, we can determine the level-spacing distribution as follows:

```
from pylab import *

# x: rescaled eigenvalues
spacings = x[1:]-x[0:-1]
hist(dat, normed=True, bins=100)
show()
```

We compare the resulting distribution with the expectation for integrable systems,



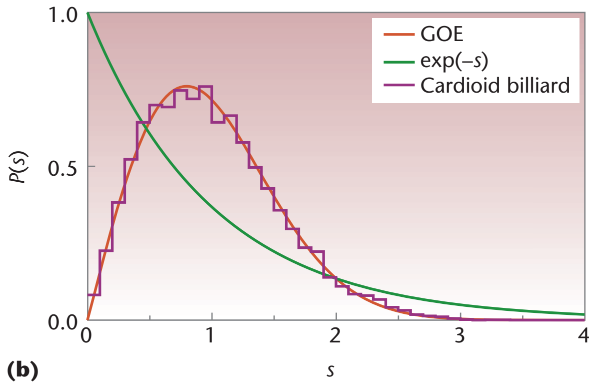


Figure 3. Level-spacing distribution. For (a) the circle billiard (100,000 eigenvalues) and (b) the cardioid billiard (11,000 eigenvalues), we find good agreement with the expected behavior of a Poissonian random process and of the Gaussian orthogonal random matrix ensemble (GOE), respectively.

$$P_{\text{Poisson}}(s) = \exp(-s). \tag{7}$$

Because $P(s) \to 1$ for $s \to 0$, this behavior is called *level attraction*. We can use the Wigner distribution to describe the level-spacing distribution for the Gaussian orthogonal random matrix ensemble (GOE) in a very good approximation[7]

$$P_{GOE}(s) \approx P_{Wigner}(s) = \frac{\pi}{2} s \exp\left(-\frac{\pi}{4} s^2\right). \tag{8}$$

In this case, we have $P(s) \to 0$ for $s \to 0$, which is called *level repulsion*. Figure 3 illustrates this behavior with a level-spacing distribution for the circle and the cardioid billiard, showing a good agreement with the expected distributions.

For the eigenfunctions of Equation 5, we would expect that they reflect the underlying classical dynamics. According to the semiclassical eigenfunction hypothesis, the eigenstates should concentrate on those regions where a generic orbit explores the long-time limit.[8] For integrable systems, motion is restricted to invariant tori, whereas the whole energy surface is filled uniformly for ergodic systems. For ergodic systems, the semiclassical eigenfunction hypothesis is actually proven by the quantum ergodicity theorem,[9] which states that almost all eigenfunctions become equidistributed in the semiclassical limit. Restricted to position space, we have

$$\lim_{j\to\infty} \int_D |\psi_{n_j}(\boldsymbol{q})|^2 \, d^2q = \frac{vol(D)}{vol(\Omega)} \tag{9}$$

for a subsequence $\{\psi_{n_j}\} \subset \{\psi_n\}$ of density one. So, for almost all eigenfunctions, the probability of finding a particle in a certain region D of the position space Ω in the semiclassical limit is just the same as for the classical system.

Figure 4 illustrates this for an integrable circle billiard and a chaotic cardioid billiard. We can clearly see that in the former case, the probability is restricted to subregions of the billiard, whereas for the ergodic case, the probability density is uniformly distributed over the full billiard region (apart from the inevitable fluctuations).

For systems with a mixed phase space, the dynamics is more complicated because regular and chaotic motion coexist (as in Figure 2). This is also reflected in the structure of quantum eigenstates, which are either located in the regular islands or extend over the chaotic region (see Figure 5). Recent results show that in certain situations this simple picture doesn't hold.[10]

Our experience with using Python for our research purposes has been extremely positive. When people think of scientific computing, typically Fortran, C, or C++ immediately come to mind, but many tasks involve fairly small amounts of time-critical code. Due to Python's efficient use of numerical libraries, no significant speed reduction arose in our applications. Moreover, all the illustrations displayed here involve Python, via PyX (http://pyx.sourceforge.net) or MayaVi (http://mayavi.sourceforge.net). Clearly, it has come into its own for many different purposes.

Acknowledgments

I thank Lars Bittrich for his useful comments on the manuscript and Bittrich, Steffen Löck, Nikolai Hlubek, and Jan-Matthias Braun for their contributions to the `Iterator` *in Figure 2.*

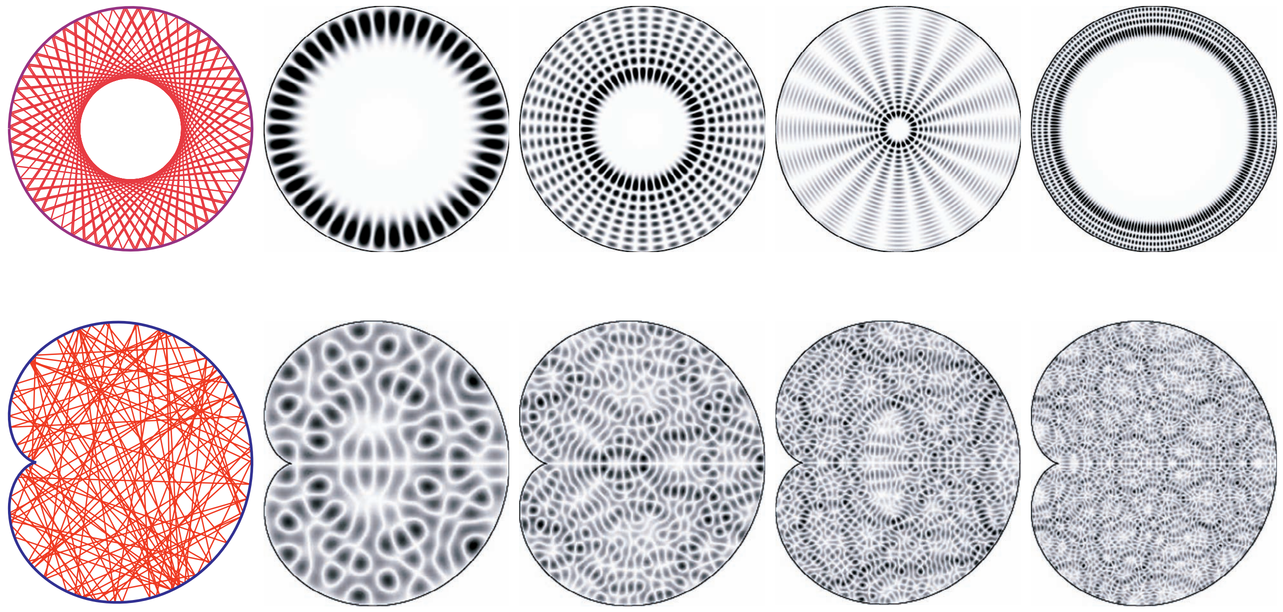


Figure 4. Behavior of eigenstates. The eigenstates of the integrable circular billiard and the chaotic cardioid billiard reflect the structure of the corresponding classical dynamics.

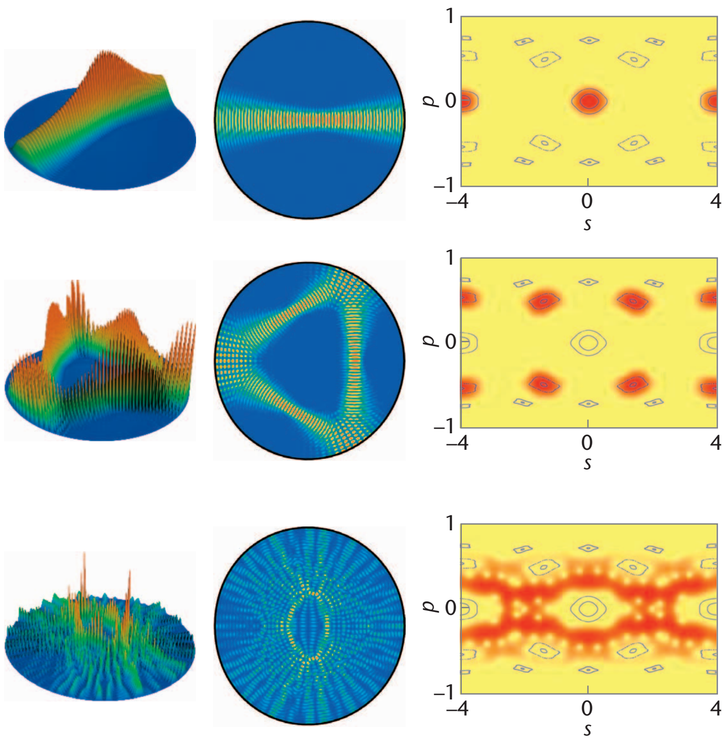


Figure 5. Mixed phase space. Eigenstates in billiards with mixed phase space typically either concentrate in the regular islands (first two lines) or extend over the chaotic region (last line). This is most clearly seen in the quantum Poincaré Husimi representation displayed in the last column for each case.

References

1. M. Robnik, “Classical Dynamics of a Family of Billiards with Analytic Boundaries,” *J. Physics A*, vol. 16, 1983, pp. 3971–3986.
2. A. Bäcker and H.R. Dullin, “Symbolic Dynamics and Periodic Orbits for the Cardioid Billiard,” *J. Physics A*, vol. 30, 1997, pp. 1991–2020.
3. A. Bäcker, “Numerical Aspects of Eigenvalue and Eigenfunction Computations for Chaotic Quantum Systems,” *The Mathematical Aspects of Quantum Maps*, Lecture Notes in Physics 618, M. Degli Esposti and S. Graffi, eds., Springer-Verlag, 2003, pp. 91–144.
4. R. Aurich and F. Steiner, “Statistical Properties of Highly Excited Quantum Eigenstates of a Strongly Chaotic System,” *Physica D*, vol. 64, 1993, pp. 185–214.
5. O. Bohigas, M.-J. Giannoni, and C. Schmit, “Characterization of Chaotic Quantum Spectra and Universality of Level Fluctuation Laws,” *Physical Rev. Letters*, vol. 52, 1984, pp. 1–4.
6. M.V. Berry and M. Tabor, “Level Clustering in the Regular Spectrum,” *Proc. Royal Soc. London A*, vol. 356, 1977, pp. 375–394.
7. M.L. Mehta, *Random Matrices*, Academic Press, 1991.
8. M.V. Berry, “Semiclassical Mechanics of Regular and Irregular Motion,” *Comportement Chaotique des Systemes Deterministes: Chaotic Behaviour of Deterministic Systems*, G. Iooss, R.H.G. Hellemann, and R. Stora, eds., North-Holland, 1983, pp. 171–271.
9. A. Bäcker, R. Schubert, and P. Stifter, “Rate of Quantum Ergodicity in Euclidean Billiards,” *Physical Rev. E*, vol. 57, 1998, pp. 5425–5447.
10. A. Bäcker, R. Ketzmerick, and A. Monastra, “Flooding of Regular Islands by Chaotic States,” *Physical Rev. Letters*, vol. 94, 2005, p. 054102.

***Arnd Bäcker** is a scientific researcher at the Technische Universität Dresden. His research interests include nonlinear dynamics, quantum chaos, and mesoscopic systems. Bäcker has a PhD in theoretical physics from the Universität Ulm. Contact him at baecker@physik.tu-dresden.de.*

An Ice-Free Arctic?
Opportunities for Computational Science

The authors discuss modeling's role in understanding the ice-ocean system, as well as its importance in predicting the future state of Arctic sea ice. In doing so, this article presents results from a hierarchy of models of different complexity, their strengths and weaknesses, and how they could help forecast the future state of the ice-ocean system.

A primary strength of 3D general circulation models (GCMs) is how well they simulate the coupled interactions between sea ice, the land surface, the atmosphere, and the ocean, all of which are essential for understanding the climate system's response to forcing perturbations. However, GCMs have limited spatial and temporal resolution (because of total integration time) and sometimes fail to capture the fundamentally important processes that affect climate variability. Moreover, the computational constraints on large models restrict the number and length of sensitivity experiments.

Component models, on the other hand, use specified forcing at the boundaries, and although they can't study the coupled system's response, they are easier to interpret and are useful for studying individual forcing parameters. Researchers can also use models of intermediate complexity, such as regional ice-ocean coupled models, to study certain processes in partially coupled modes. Perhaps the best option of all is to use a hierarchy of models—a combination of intermediate-complexity models, process models, and GCMs—to gain a clearer understanding of how multiple processes can affect, say, the high-latitude climate system.

The field of climate variability involves a wide range of spatial and temporal scales. Small spatial-scale processes such as turbulence, mixing, and convection, for example, affect large-scale ice-ocean-atmosphere circulation patterns, which determine the system's basic state, which in turn affects small-scale processes. Small spatial-scale processes also typically operate over shorter timescales. Resolving (or parameterizing) the climate system's smaller-scale features while performing long-term integrations on complex GCMs constitutes the principal challenge for computational scientists interested in the field.

In this article, the authors discuss future projections of the Arctic sea-ice cover from sophisticated GCMs, the uncertainties associated with these projections, and how the use of simpler component models can help in the interpretation of complex GCMs.

1521-9615/07/$25.00 © 2007 IEEE
Copublished by the IEEE CS and the AIP

L. Bruno Tremblay
McGill University
Marika M. Holland
US National Center for Atmospheric Research
Irina V. Gorodetskaya
Columbia University
Gavin A. Schmidt
NASA Goddard Institute

Ice-Ocean Modeling

By Uma Bhatt and David Newman, University of Alaska Fairbanks

This issue's article for the International Polar Year focuses on various methods for modeling ice-ocean interactions. This is timely not just because of the IPY but also because of the much publicized shrinking Arctic ice cap and expected changes in climate due to shifts in ocean currents.

The following Web sites highlight different aspects of a changing Arctic from satellite data to model projections and intercomparisons:

- NASA's Scientific Visualization Studio site is a great place to start because it plays movies created from satellite measurements as well as from model projections (http://svs.gsfc.nasa.gov/). To see beautiful animations of sea-ice changes, search for "sea ice" on this site; one of the best depicts the minimum ice concentration from 1979 to 2006 (http://svs.gsfc.nasa.gov/vis/a000000/a003300/a003378/). For a variety of other movies of Arctic data, go to http://svs.gsfc.nasa.gov/search/Keyword/Arctic.html.
- Scientists at the Geophysical Fluid Dynamics Laboratory (GFDL; www.gfdl.noaa.gov) are investigating climate variability and prediction from annual to centennial timescales. You can see one of their state-of-the-art models of the shrinking Arctic ice cap at www.gfdl.noaa.gov/research/climate/highlights/GFDL_V1N1_gallery.html.
- The US National Center for Atmospheric Research is home to the Community Climate System Model (www.ccsm.ucar.edu); as the name indicates, the climate community is heavily involved in the model's development. You can find an overview of the model at www.ucar.edu/communications/CCSM and more about high-latitude simulations at www.ccsm.ucar.edu/working_groups/Polar.
- To make more sense of the results of different models worldwide, Lawrence Livermore National Laboratory has established a program to facilitate model comparison (www-pcmdi.llnl.gov/projects/cmip/).

The article in this issue describes two extremes in the hierarchy of models used to investigate ice-ocean interactions—namely, large-scale global models and small-scale ice models. In between these is a class of models called *regional models*, which are typically forced with either real climate data or GCM data at their boundaries and can be run at higher resolution to investigate smaller-scale effects, such as local orography, smaller-scale weather forcing effects, and so on. You can find more information on the intercomparison of these arctic regional ice-ocean models at the Arctic Ocean Model Intercomparison Project (AOMIP; http://fish.cims.nyu.edu/project_aomip/overview.html).

The next article in our series dedicated to the IPY will move onto land and provide insights into modeling high-latitude terrestrial vegetation dynamics, once again using a hierarchy of models of varying complexity. Of course, due to space constraints, we can't cover all the relevant topics in this series, and most notable among our omissions is coverage of biogeochemical processes. One of particular relevance for the polar oceans is the carbon cycle in the ocean; recent studies show that the acidification of the ocean due to enhanced carbon dioxide is particularly important in the cold polar waters. This acidification is expected to dissolve the calcium-based shells of small marine organisms, unleashing a major impact on the food chain. Models of these chemical-biological processes are at an early stage of development, although researchers expect biogeochemistry models to become an integral part of what are presently classified as climate models.

Arctic Sea Ice

Over the past few decades, the Arctic has witnessed large changes in its land, atmosphere, ice, and ocean components. These changes include a decrease in sea-ice extent and thickness,[1,2] a warming of surface air temperatures,[3] a decrease in the sea-level pressure,[4] deeper penetration of storms in the eastern Arctic,[5] a warming of the North Atlantic drift current and its flow at depth beneath the fresher Arctic surface waters (see Figure 1),[6] the melting of the permafrost,[7] increased river runoff,[8] and changes in vegetation,[9] among others. Of all these changes, the best documented is the decrease of the minimum sea-ice extent as observed by satellite (see Figure 2). Scientists have seen a decrease in September sea-ice extent of 8 percent per decade since the late 1970s,[10] with three minimum ice records broken in the past four years.

All these observations are internally consistent with local feedbacks from the ice, clouds, and the surface energy budget (the balance of energy coming in and then leaving the surface). At high latitudes, the dominant feedback mechanism believed to be responsible for increased local warming is called *ice-albedo feedback*. If the climate warms, the sea ice in the polar seas retreats, and the fraction of solar radiation absorbed by the ice-ocean system increases (sea ice reflects most of the incoming solar radiation; the ocean absorbs most of it). This leads to further warming of the ocean surface and the overlying air, further retreat of the sea ice, further warming, and so on. This positive feedback can cause large and very rapid changes in surface conditions and local climate—early models based on sea-ice albedo feedback alone predicted several degree changes in the global mean temperature initiated by small changes in radiative forcing.[11]

In the real world (and in more complex models), negative feedback mechanisms damp or delay the climate system's response to changes in forcing. One such mechanism operating at high latitudes is the *cloud-albedo feedback*. When sea ice retreats, more ocean water is exposed to the atmosphere. This leads to more evaporation (the overlying warmer air can hold more water vapor than the colder atmosphere) and potentially more clouds, which are highly reflective of solar radiation. In effect, we've replaced a highly reflective material at the surface (sea ice) with an equally reflective surface up in the atmosphere (the clouds), but they don't cancel each other out entirely. Instead, the combined effect of changes in cloud and sea ice has a reduced but still significant effect on top-of-atmosphere (TOA) albedo, which is important to global temperature (see Figure 3). In fact, in a cloudier Arctic, increased longwave radiation reaching the surface can intensify sea-ice melt, especially during the spring.[12]

In the late 1980s and early 1990s, more storms than usual penetrated deep into the eastern Arctic, a phenomena that became part of a trend in the North Atlantic Oscillation (NAO). During a positive NAO phase, storms preferentially move northward in the Icelandic and Barents Seas (rather than across the Atlantic or Baffin Bay), with the sea-level pressure in the northern part of the North Atlantic relatively lower. These storms carry sensible and latent heat north, create wind patterns that blow ice away from the coastlines of the Kara and Laptev Seas,[13] and export thick multiyear ice from the central Arctic through the Fram Strait,[14] which thins the ice in the peripheral seas. The associated heat flux from the relatively warm ocean through the thin ice cover keeps the overlying atmosphere warmer. These storms also result in a greater poleward heat transport (both in the ocean and in the atmosphere), warmer surface air temperature in the eastern Arctic, deeper penetration of North Atlantic drift waters along the continental shelf, less multiyear ice in the central Arctic, and increased precipitation and runoff from the Eurasian continent.

All the feedback mechanisms we mentioned earlier lead to a larger warming signal—called *polar amplification*—in the high latitudes, particularly in the northern hemisphere, which has a perennial sea-ice cover and the potential for a stronger ice-albedo feedback signal. As a result, although climate models predict a global mean warming of 3 to 5 degrees Celsius by the end of the 21st century (assuming a continued increase in greenhouse gas[15]), the same models predict a warming of 10 to 15 degrees Celsius and a much reduced sea-ice cover in the Arctic for the same time frame.[16] Because of polar amplification, the Arctic region could be a place where scientists can more clearly separate a warming signal associated with human

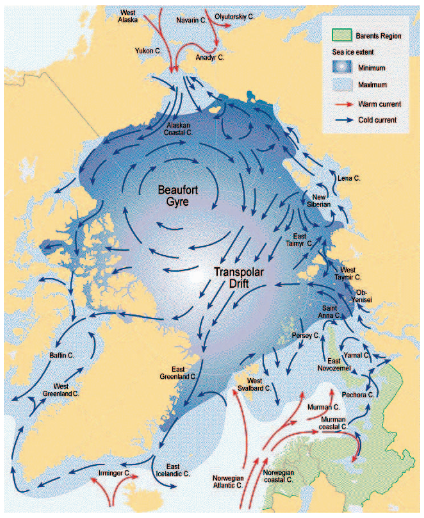


Figure 1. Arctic Ocean surface circulation. Red arrows indicate warm Atlantic Ocean currents and blue arrows indicate cold Arctic surface currents. North Atlantic drift waters entering the Arctic west of Svalbard flow counterclockwise at depth (the warm core is at roughly 300 meters) and exit through the Fram Strait (not shown).

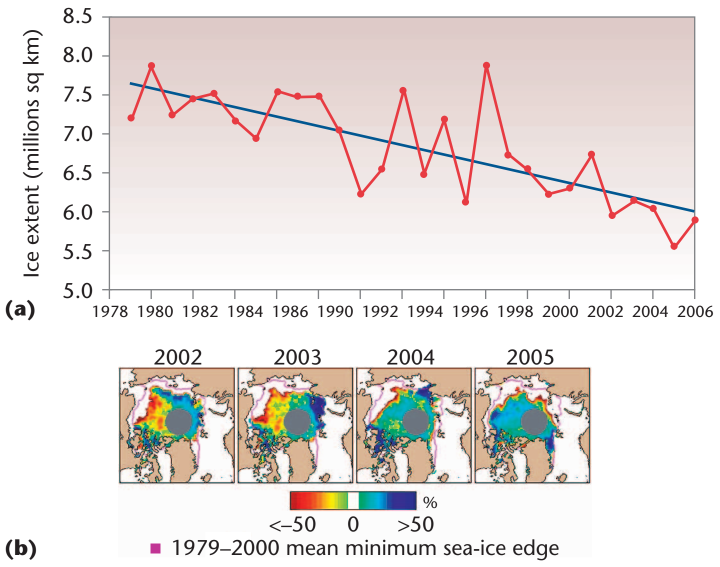


Figure 2. Satellite observation. (a) The trend in September sea-ice extent in the Arctic, and (b) sea-ice extent anomalies for 2002, 2003, 2004, and 2005. The pink line represents the mean ice-edge position averaged over the satellite era.

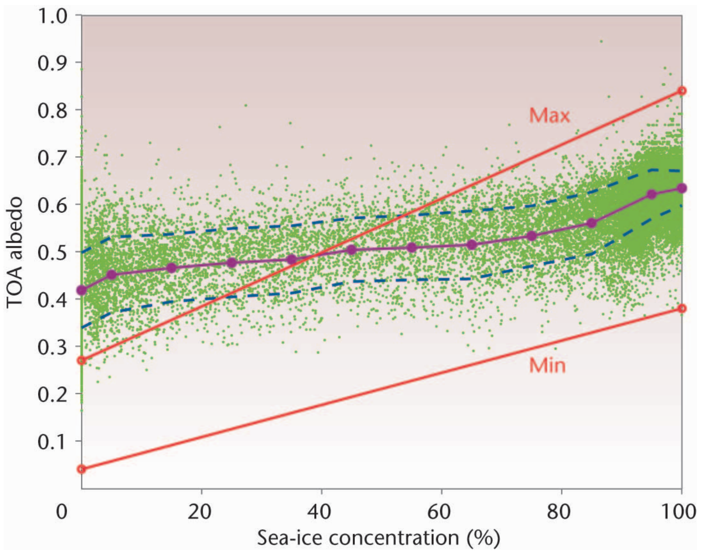


Figure 3. Monthly mean top-of-atmosphere (TOA) albedo (green dots) against northern hemisphere sea-ice concentrations (SICs). Purple dots connected with a line represent area-weighted TOA albedo averages for each 0, 100, and 10 percent bin of SIC. The dashed lines are standard deviations, and the thin red lines connect maximum and minimum observed surface albedo values for both the open ocean and sea ice, corresponding to the TOA albedo envelope in the absence of an atmosphere. (The TOA albedo is from the Earth Radiation Budget Experiment's [http://asd-www.larc.nasa.gov/erbe/ASDerbe.html] data, and the SICs are from the Hadley Centre's sea-ice and sea-surface temperature data set [http://hadobs.metoffice.com].[23])

activity from naturally occurring climate variability. These are among the reasons why climate scientists are interested in monitoring and modeling the high latitudes, and why they do field work in such remote and harsh environments.

The Ice-Ocean System's Mean State

Sea ice and oceans are present in both hemispheres at high latitudes. Yet, the two systems' natures and behaviors are very different.

Sea ice forms when surface waters reach their freezing points (roughly –1.8° Celsius for typical ocean waters), but for this to occur, the surface ocean must be stratified.[17] When seawater cools, it becomes denser—the heavier surface waters sink (*convect*) and mix with deeper waters. In a stratified ocean, the depth to which the water convects is limited to the surface layer, so only the first few tens of meters (roughly 40 meters, for the Arctic) must cool to the freezing point for sea ice to form. In an unstratified ocean, the entire water column (roughly 4,000 meters) would need to cool before ice could form on the surface.

In the Arctic, surface stratification mainly comes from the input of fresh water from river runoff. The Arctic Ocean constitutes 2 percent of the Earth's total ocean volume, yet it receives approximately 10 percent of total continental runoff. The cold and relatively fresh surface waters (the mixed layer) sit above an equally cold but somewhat saltier layer of water called the *cold halocline layer* (CHL). Beneath this layer are warmer and saltier waters from the North Atlantic. The CHL's presence in the Arctic buffers the cold surface waters from the warmer Atlantic waters by limiting the ocean heat flux into the mixed layer during the winter growth season, which in turn helps the buildup of a perennial sea-ice cover. Two different mechanisms explain the CHL's formation. In the first, shelf-water advection feeds the cold halocline waters: when ice forms at the beginning of the cold season it rejects salt, making the relatively fresh shelf waters saltier. These waters are advected offshore and find their level of equilibrium between the lighter (fresher) surface waters and the heavier (saltier) Atlantic waters. In the second, deep-ocean convection feeds the cold halocline waters; the relatively fresh shelf waters are advected offshore and remain near the surface (http://psc.apl.washington.edu/HLD/Lomo/Lomonosov.html).

In the Southern Ocean, surface stratification is much weaker and is mainly due to melting ice shelves, melting sea ice, and the runoff from the continental ice sheet. The mixed layer sits directly atop the warmer and saltier pycnocline waters. In early winter, once the shallower seasonal pycnocline (from the previous summer's sea-ice melt) is eliminated when ice grows in fall, further ice formation (and salt release) later in the winter causes convection and entrainment of the warmer sub-pycnocline waters to the surface. This will melt or prevent from forming approximately 1.5 meters of ice each winter.

A Seasonally Ice-Free Arctic

When scientists talk about an ice-free Arctic, they're generally referring to a summer ice-free Arctic Ocean—that is, one that has lost its perennial sea-ice cover, a situation that's sometimes called the *Antarctic analogue*.[18] In winter, no model projects a complete disappearance of the sea-ice cover until at least the end of this century.

In the Arctic Ocean, approximately 1 meter of ice forms each year during winter, 0.7 meters melt during the summer, and an equivalent of 0.3 meters are exported south to the North Atlantic where it melts. We could achieve a seasonally ice-free Arctic through a sustained increase in sea-ice export out of the Arctic via the Fram Strait,[14] a decline in winter sea-ice production, or an increase in summer sea-ice melt.

Anomalous Ice Export

The mean time sea ice resides in the Arctic is approximately seven years. For the sea-ice export to have a significant impact on the volume of ice remaining in the Arctic, anomalous wind patterns must be maintained for at least this amount of time. However, as we mentioned earlier, a strong negative feedback limits the impact of enhanced sea-ice export on Arctic ice volume.[14]

When export is anomalously high, the volume (or mean thickness) of ice left behind decreases, and the heat lost from the ocean to the atmosphere (and concomitant sea-ice formation) increases. In the late 1980s and early 1990s, researchers observed a trend toward a more positive NAO index, with deeper penetrations of storms in the eastern Arctic and winds blowing the thick multiyear ice from the central Arctic out through the Fram Strait. Some scientists have hypothesized that the very low ice observed in subsequent years is the result of this trend.[3] However, since the mid-1990s, the NAO index hasn't been as positive, yet the system hasn't recovered.

Anomalous Winter Sea-Ice Growth

The typical heat loss from the Arctic Ocean to the atmosphere is 15 Watts per square meter (W m^{-2}), which is equivalent to a 1-meter ice growth over an 8-month growing season. In winter, the dominant factors in the surface heat balance are upwelling and downwelling longwave radiation and the conductive heat flux through the sea ice.[19] On a typical clear-sky day, the net longwave radiation emitted from the surface is approximately 30 W m^{-2}, whereas the net longwave radiative flux drops to almost zero during cloudy skies.

The expected increase in downwelling longwave radiation by 2050, as predicted by the latest generation of GCMs from the International Panel on Climate Change's 4th Assessment (IPCC-AR4), ranges from roughly 10 to 25 W m^{-2}, depending on the model used and the future CO_2 increase scenario considered. This is significantly larger than the same models' global average, which ranges from 3 to 15 W m^{-2}, and is of the same order of magnitude as the net heat loss to the atmosphere during the winter months. An increase in the downwelling longwave radiation will result in a warmer surface-ice temperature, a reduced temperature gradient from the ice base to its surface, and a reduced winter ice growth. These changes would gradually decrease the winter sea ice's growth over time, if no other feedback mechanisms were present. Of all the IPCC models participating in the 4th assessment that have a realistic seasonal ice extent cycle, 40 percent display this gradual decrease.[20]

Anomalous Summer Sea-Ice Melt

In summer, the main balance in the Arctic sea ice's surface heat budget is between the net shortwave radiation absorbed at the surface, the energy required to melt the sea ice, and to a lesser extent the net longwave radiation lost by the surface.[19] Clouds have a large impact on surface melt as well. Whereas winter clouds have a warming effect (increased downwelling longwave radiation), summer clouds reduce the amount of shortwave radiation that reaches the surface and typically have a cooling effect (the increased downwelling longwave radiation associated with clouds doesn't compensate for the decreased downwelling shortwave radiation[21]). Depending on microphysical properties (such as cloud-particle radius and ice versus liquid), clouds can affect the surface radiation balance differently in winter and summer.[22,23]

How the summer melt will change in response to future greenhouse gas production depends largely on the projected changes in Arctic cloud cover and type. At present, satellite observations show an increase in the melt season by a few weeks, associated with the NAO's more positive phase,[10] and possibly with an increase in the downwelling longwave radiation reaching the surface.[12]

Feedback Mechanisms

Increased ice export, decreased winter growth, and increased summer melt will all result in a gradual change in sea-ice conditions in the Arctic Ocean. Let's examine how slowly varying CO_2 increases in the atmosphere could lead to a rapid decline in Arctic sea-ice volume if we reach certain thresholds in sea-ice thickness or surface ocean temperature and salinity structure.

Dynamic Feedback

Energy input by the wind dissipates due to both bottom friction between the ice base and ocean surface and lateral friction between ice floes rubbing against one another along shear lines. When local convergence is present,[24] ridges form, leading to an increase in the system's potential energy. A thinner sea-ice cover has a lower mechanical strength and deforms more easily (sea ice compressive and shear strengths are functions of thickness, but sea ice tensile strength is invariably much lower because the pack ice is a highly fractured

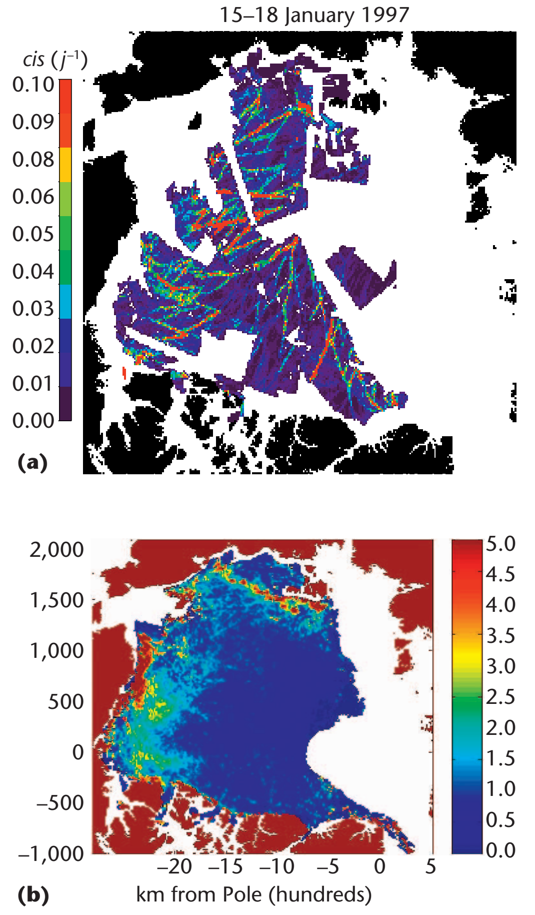


Figure 4. Lead pattern in Arctic sea ice. (a) Shear deformation from the Radarsat Geophysical Processor System (RGPS; www-radar.jpl.nasa.gov/rgps/radarsat.html); (b) mean spatial distribution of open water over the Arctic Ocean for the winters of 1996–1999, derived from RGPS thin ice thickness data. Open water is defined as the ice surface area covered by ice thinner than 10 cm.

material that can't sustain large tensile stresses before deforming). Moreover, faster-moving ice yields more mixing at the surface.

Localized high shear deformation can also raise the pycnocline depth (due to Ekman upwelling) and increase turbulent heat fluxes in the surface ocean boundary layer.[25] Figure 4 shows typical lead (a narrow opening in the sea-ice pack that exposes the open ocean) patterns as well as the spatial distribution of leads from three years of Radarsat Geophysical Processor System (RGPS) data. A faster-moving sea-ice cover has a shorter life span in the Arctic (with wind forcing remaining the same), which results in decreased thermodynamic growth.

Loss of the Cold Halocline Layer

As discussed earlier, the CHL buffers the Arctic's cold surface water from the warm Atlantic layer beneath. In the early 1990s, Michael Steele and Timothy Boyd[26] showed that the eastern-central Arctic CHL was weak in 1993 and completely absent in 1995. Without a CHL, the Arctic Ocean assumes characteristics of the Antarctic water column, and with that, presumably an increased ice-ocean heat exchange and a behavior similar to the Antarctic ocean-ice system (that is, a seasonal ice cover). In the late 1990s, the CHL returned.[27] Researchers argued that this excursion was due to a change in the large-scale atmospheric circulation and a concomitant change in river inflow paths along the Eurasian shelf.[28] During that time, the river runoff from Eurasia formed an eastward-flowing coastal current in response to large-scale wind pattern changes (as opposed to flowing off the shelf and along the Lomonosov ridge in the central Arctic).

It's unclear how the CHL will respond to reductions in ice cover. Weakening or loss of the CHL would constitute a large positive feedback mechanism accelerating the decline of Arctic sea-ice cover. To quantify the amount of heat brought up to the surface when sea ice forms, rejects salt, and enables surface convection, Douglas Martinson and Richard Iannuzzi[29] developed a simple bulk model based on the upper ocean's temperature and salinity profile. Douglas Martinson and Michael Steele[18] later calculated (using all available temperature and salinity profiles from the Arctic) the latent ocean heat fluxes that would be released in the event of a CHL loss. The values of heat fluxes ranged from 17 W m^{-2} north of Greenland in the Amundsen Basin to approximately 9 W m^{-2} in the Canadian Basin. Given that the CHL's presence is linked with river runoff paths into the Arctic Ocean and shelf water's hydrographic properties, whether we could lose the CHL over the entire Arctic Ocean at once remains an open question. However, even a partial loss over a limited region of the Arctic would significantly impact the sea-ice cover's thinning.

Ice-Ocean-Albedo Feedback

Another possible mechanism for the rapid decline in Arctic sea-ice cover is linked with *ice-ocean-albedo feedback*. As the sea ice gradually thins due to increased greenhouse gases in the atmosphere, we'll reach a threshold when an anomalously warm year (associated with natural interannual variability) causes a significant increase in open water. This will be followed by increased absorption of solar radiation in the mixed layer and an increase in basal melt along with the usual surface melt. The natural variability that can trigger these events includes higher than normal atmospheric and oceanic heat fluxes to the Arctic.[20] Observations taken along the Eurasian continental shelf show a pulse-like warming of the Atlantic water circulating cyclonically along the shelf in the Arctic Ocean, and scientists also recorded a few warm events of roughly 1° Celsius in 1990[30] and 2004.[6] Researchers simulated similar warming events with a regional ice-ocean model forced with specified atmospheric forcing.[31]

The Arctic Sea-Ice Cover's Future

The models participating in the IPCC's 4th assessment represent the state of the art in global climate modeling. They incorporate ocean, atmosphere, terrestrial, and sea-ice components as well as the linkages among them.

IPCC-AR4 models consistently exhibit a decrease in Arctic sea-ice cover in response to increasing greenhouse gases, but this retreat ranges widely in its rate and magnitude. Some of this scatter is related to the simulated present-day climate conditions; models with more extensive ice cover in the current climate tend to be less sensitive. An analysis of 14 IPCC-AR4 models shows that the retreat of sea ice by the mid-21st century is correlated to late 20th century conditions with a correlation coefficient of $R = 0.42$, where R stands for the correlation coefficient between the simulated sea-ice extent in the middle of the 21st century and those of the late 20th century. By the end of the 21st century, however, the correlation degrades to $R = 0.09$, such that, although initial ice conditions are important, other processes dominate. These processes affect climate-feedback strength and can include simulated cloud cover, atmospheric circulation's meridionality (the north–south heat-moisture transport), and the mean and variability of ocean heat transport to the Arctic.

The mechanism for losing perennial sea-ice cover is thermodynamic in nature—that is, it's due to an increased net surface and basal heat budget and melt. For the multimodel ensemble mean, the ice melt for the 2040–2060 average increases over that of the 1980–2000 average by 1.2 meters, whereas the net ice growth increases by 0.2 meters and the ice export decreases by 0.1 meters. This clearly shows that the dominant term for the ensemble mean is sea-ice melt, with ice export and winter growth acting as negative feedbacks. However, the variability in winter growth and ice export is larger, and, for some models, they act as positive feedbacks. Of all the processes that could be responsible for Arctic sea-ice decline, increased summer melt is thus the main player.

Limitation of Current GCMs

We noticed major improvements in Arctic simulation quality from the latest generation of models participating in the IPCC's 4th assessment when compared to the previous generation of models. These include the representation of sea-ice thickness distribution, the simulation of sea-level pressure at high latitude, the atmospheric circulation pattern's meridionality, and precipitation patterns at high latitudes (a discussion of Arctic climate biases associated with the previous generation of GCMs appears elsewhere[32]). Important issues remain, however, and they'll require further attention before we can rely entirely on model predictions of the Arctic's future climate.

Clouds, for example, are inherently difficult to model because of the small-scale nature of the processes that govern their formation. Moreover, Arctic clouds are difficult to measure remotely because distinguishing them from the sea ice surface in infrared and visible satellite images is difficult. The lack of data and difficulty in collecting it also poses a challenge to the study of Arctic clouds. Fortunately, several cloud detection algorithms specific to the polar regions have been developed recently and validated against ground-based campaigns, whereas the newly launched satellite programs have much improved capabilities in differentiating clouds from sea ice and quantifying the clouds' microphysical properties (such as cloud ice and liquid water content and particle size).[33,34]

Measurements conducted during the Surface HEat Budget of the Arctic (SHEBA; http://sheba.apl.washington.edu) experiment showed that liquid water dominates cloud water content over the ice phase in summer, and even winter clouds contain significant amounts of liquid water.[21] Moreover, liquid clouds reflect more shortwave radiation, whereas ice clouds are relatively more transparent. The model parameterizations that researchers use to decide whether a cloud is liquid or solid are simple and often based on relatively few field campaigns that aren't always applicable to Arctic conditions. Figure 5 shows the partitioning between liquid and ice in the Arctic from SHEBA observations and three coupled models participating in the IPCC's 4th assessment report. Models with the largest liquid water content show the smallest downwelling shortwave flux during the summer, which is partly compensated for with an increased longwave flux. During the winter months, models with liquid-dominated clouds have higher downwelling longwave radiation compared to models with ice-dominated clouds.

Another small-scale process that isn't well resolved in current GCMs is linked with determin-

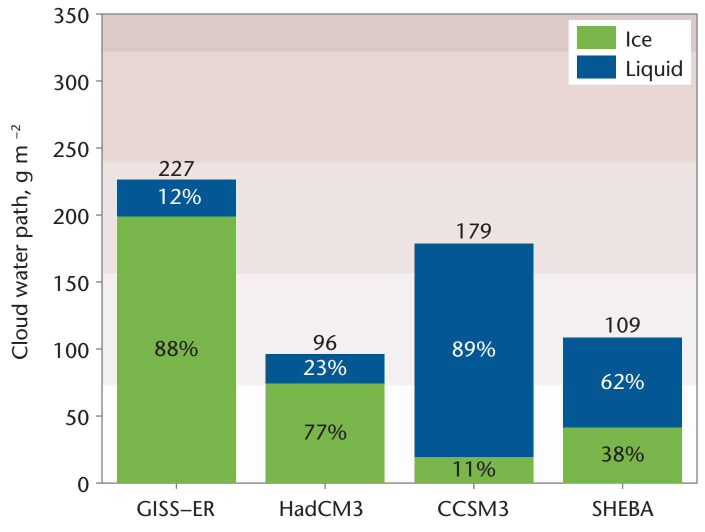


Figure 5. Liquid vs. ice clouds. In comparing the May–September cloud ice vs. liquid water paths from SHEBA's ground-based observations and the GISS-ER, HadCM3, and CCSM3 models, we see a dominance of liquid water clouds in summer, which is sometimes underestimated in some models. The data are averaged from 1959–1998; the numbers above each bar indicate the total cloud water paths (g m^{-2}), and the percentages show the partitioning into liquid and ice phases.

ing the upper ocean's vertical structure—in particular, the CHL. GCMs often have problems when resolving the sharp salinity gradients at the base of the mixed layer; instead, they tend to produce saltier surface waters and fresher pycnocline waters, which results in a warm Atlantic layer that isn't buffered from the surface. However, the warm Atlantic layer is often deeper than observed because of the upper water column's salinity structure (see Figure 6). The two effects compensate for each other, often giving realistic sea-ice heat and mass balance. Whether the variability around the mean or the response in a changing climate is realistic remains an open question.

A Simple Modeling Approach

To separate the effect of anomalous atmospheric circulation, increases in the downwelling longwave radiation associated with increased greenhouse gas, and the loss of the CHL in creating a summer ice-free Arctic, we can use a simple stand-alone viscous plastic sea-ice model coupled to a slab ocean and atmospheric energy balance model (a detailed description of the model appears elsewhere[35]). To this end, we ran the model continuously with 1989 wind forcing because that year had an anomalously high NAO index and sea-ice export; with an increased downwelling longwave radiation of 20 W m^{-2}, a typical value simulated by IPCC-AR4 models for 2050; and with a specified ocean heat flux of 20 W m^{-2}, mimicking the loss of the CHL in the Arctic Ocean. Figure 7 shows a present-day climate simulation. We forced the model run with atmospheric forcing fields for the 1949–2005 time period. In our sensitivity studies, we modified only one forcing field at a time; the other fields remained the same as for the present climate run. In all cases, we used a 10-year mean sea-ice thickness field, calculated from the past 10 years of a 50-year run.

The main features of the present-day climate stand-alone model simulation include thicker ice north of the Canadian Arctic Archipelago (5 to 6 meters) and thinner ice along the Eurasian Basin (1 meter), in good agreement with observations from submarine and satellite altimeter estimates (http://nsidc.org). This sea ice thickness pattern is due to the dominant winter winds that tend to push ice from the Eurasian continent toward North America as well as the longer life span of sea ice caught in the Beaufort Gyre. The asymmetry in ice thickness is particularly interesting: ice is thicker in the Beaufort Gyre than in the Lincoln Sea (north of Greenland) because of the advection by the Beaufort Gyre of thick multiyear ice westward in front of the Canadian coastline.

When forcing the model with continuous 1989 wind forcing (and keeping everything else the same), the steady-state response (achieved after seven years) results in a change in sea ice thickness, with thinner ice primarily in the East Siberian Sea and the Beaufort Gyre (see Figure 7c). The export of thick multiyear ice from the Lincoln Sea is also clearly visible in the Fram Strait and along the East Greenland coastline. In contrast, the simulation with increased downwelling longwave radiation results in much thinner ice over the entire Arctic Ocean (see Figure 7b). Of the three effects that scientists believe have the biggest impact on sea-ice cover, the loss of the CHL is greater because it reduces winter ice growth and contributes to a thinner end-of-winter sea-ice thickness that's more prone to substantial summer melt.

The timescale associated with the decline of sea-ice cover in these simulations also differs. Changes in the longwave forcing are gradual and occur over longer timescales than the ice-surface-ocean system's steady-state response (roughly seven to eight years). The sea-ice cover's response in this simulation is therefore in equilibrium with the forcing. On the other hand, both the changes in large-scale atmospheric circulation and in the CHL can occur on much shorter timescales, so the ice-surface-ocean response time governs the system's response, which in turn leads to rapid changes in sea-ice conditions. For this reason, while the magnitude of the surface forcing for both the increased downwelling longwave radiation and the loss of the CHL is of the same order of magnitude, a loss of the CHL could lead to a much more rapid decline of the sea-ice cover.

The study of polar oceans, sea ice, and the high-latitude climate relies heavily on regional models and GCMs that incorporate several critical Earth system components. Climate models suggest a transition to ice-free Arctic conditions in the summer in the near future (in 50 to 100 years). This represents an unprecedented change in the Arctic climate, with potentially far-reaching effects.

Fortunately, several institutions, including national research centers and universities, have groups of researchers working on the development, numerical implementation, and coupling of new and improved climate models. These group efforts provide a unique opportunity for scientists in the computational sciences to tackle important climate issues in a stimulating multidisciplinary research environment.

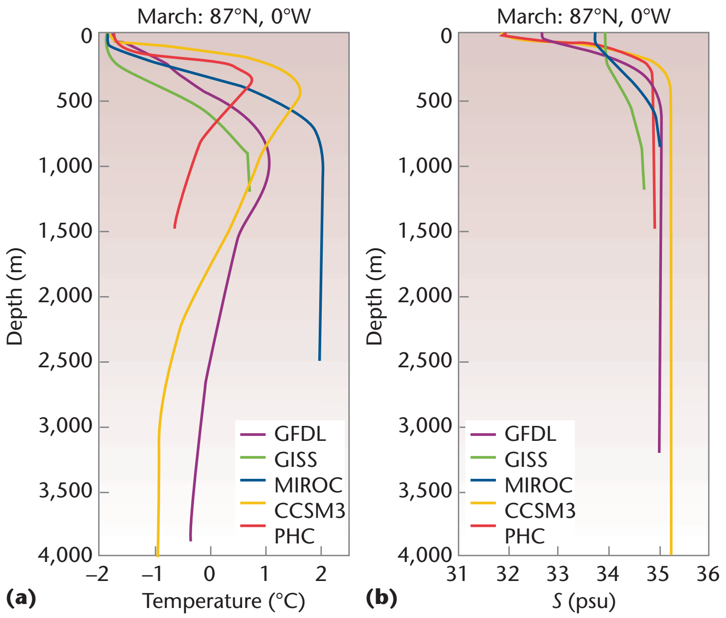


Figure 6. Vertical ocean temperature and salinity structure. Four models give different (a) simulated March temperatures and (b) vertical salinity profiles. Of particular interest is the fact that most models have a reduced stratification in the upper ocean and a warm Atlantic layer deeper than observed. The exception is CCSM3, which has a good representation of surface stratification, even though it shows Atlantic waters as 1° Celsius too warm and somewhat saltier.

Acknowledgments

This survey was supported by the National Science and Engineering Research Council of Canada's Discovery Grant program and the US National Science Foundation under grants OPP-0230264, OPP-0230325, and ARC-05-20496. We thank the modeling groups for providing their data for analysis, the Program for Climate Model Diagnosis and Intercomparison (PCMDI) for collecting and archiving the model output, and the Joint Scientific Committee/Climate Variability and Predictability Working Group on Coupled Modeling (WGCM) for organizing the model data analysis activity. The US Department of Energy's Office of Science supports the data archive.

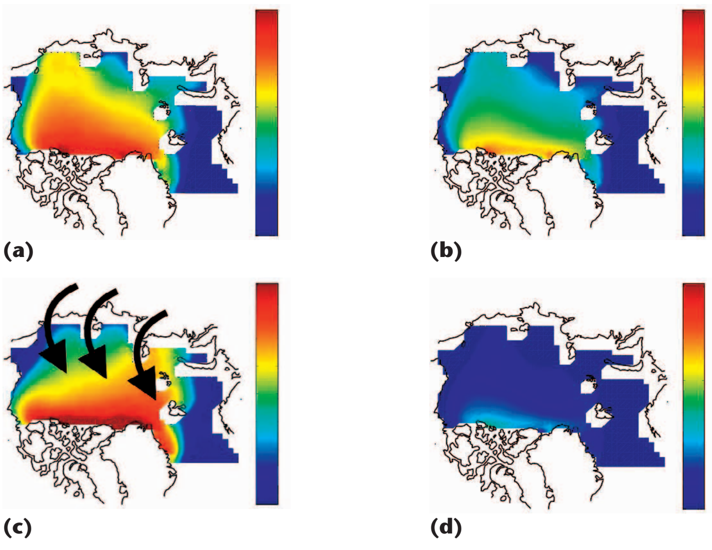


Figure 7. Sensitivity studies using the stand-alone model. We simulated the 10-year mean September sea-ice thickness distribution from a stand-alone granular sea-ice model forced with (a) climatological forcing, (b) increased downwelling longwave radiation, (c) continuous daily varying 1989 wind forcing, and (d) an increased ocean heat flux (17 W m^{-2}). The black arrows show the eastern Arctic's dominant winter wind pattern in 1989.

References

1. D.J. Cavalieri et al., "Observed Hemispheric Asymmetry in Global Sea Ice Changes," *Science*, vol. 278, no. 5340, 1997, pp. 1104–1106.
2. D. Rothrock, Y. Yu, and G. Maykut, "Thinning of the Arctic Sea-Ice Cover," *Geophysical Research Letters*, vol. 26, no. 23, 2000, pp. 3469–3472.
3. I.G. Rigor et al., "Variations in Surface Air Temperature Observations in the Arctic," *J. Climate*, vol. 13, no. 5, 2000, pp. 896–914.
4. J.M. Walsh, W.L. Chapman, and T.L. Shy, "Recent Decrease of Sea Level Pressure in the Central Arctic," *J. Climate*, vol. 9, no. 2, 1996, pp. 480–488.

5. G.J. McCabe, M.P. Clark, and M.C. Serreze, "Trends in Northern Hemisphere Surface Cyclone Frequency and Intensity," *J. Climate*, vol. 14, no. 12, 2001, pp. 2763–2768.
6. I.V. Polyakov et al., "One More Step Toward a Warmer Arctic," *Geophysical Research Letters*, vol. 32, 2005; doi:10.1029/2005GL023740.
7. L. Hinzman et al., "Evidence and Implications of Recent Climate Change in Northern Alaska and Other Arctic Regions," *Climatic Change*, vol. 72, no. 3, 2005, pp. 251–298.
8. B.J. Peterson et al., "Increasing River Discharge to the Arctic Ocean," *Science*, vol. 298, no. 5601, 2002, pp. 2171–2173.
9. M. Sturm, C. Racine, and K. Tape, "Climate Change: Increasing Shrub Abundance in the Arctic," *Nature*, vol. 411, no. 6837, 2001, pp. 546–547.
10. J.C. Stroeve et al., "Tracking the Arctic's Shrinking Ice Cover: Another Extreme September Minimum in 2005," *Geophysical Research Letters*, vol. 32, no. 4, 2004; doi:10.1029/2004GL021810.
11. M.I. Budyko, "The Effects of Solar Radiation on the Climate of the Earth," *Tellus*, vol. 21, no. 611, 1969, pp. 611–619.
12. J.A. Francis et al., "Clues to Variability in Arctic Sea Ice Minimum Extent," *Geophysical Research Letters*, vol. 32, no. 21, 2005.
13. A.E. Armstrong, L.-B. Tremblay, and L.A. Mysak, "A Data-Model Intercomparison Study of Arctic Sea-Ice Variability," *Climate Dynamics*, vol. 20, no. 5, 2003, pp. 465–476.
14. G. Arfeuille, L.A. Mysak, and L.-B. Tremblay, "Simulation of the Interannual Variability of the Wind-Driven Arctic Sea-Ice Cover during 1958–1998," *Climate Dynamics*, vol. 16, Feb. 2000, pp. 107–121.
15. J.T. Houghton et al., *Climate Change 2001: The Scientific Basis*, Cambridge Univ. Press, 2001.
16. M.M. Holland and C. Bitz, "Polar Amplification of Climate Change in Coupled Models," *Climate Dynamics*, vol. 21, Sept. 2003, pp. 221–232.
17. K. Aagaard and E.C. Carmack, "The Role of Sea Ice and Other Fresh Water in the Arctic Circulation," *J. Geophysical Research*, vol. 94, Oct. 1989, pp. 14485–14498.
18. D.G. Martinson and M. Steele, "Loss of the Arctic Cold Halocline: Is the Arctic becoming More Like the Antarctic?," *Geophysical Research Letters,* vol. 28, no. 2, 2001, pp. 307–310.
19. H. Huwald, L.-B. Tremblay, and B. Blatter, "Reconciling Different Datasets from the Surface Heat Budget of the Arctic (SHEBA)," *J. Geophysical Research*, vol. 110, May 2005; doi:10.1029/2003JC002221.
20. M.M. Holland, C. Bitz, and L.B. Tremblay, "Future Abrupt Reductions in the Summer Arctic Sea Ice," *Geophysical Research Letters*, vol. 7, Dec. 2006.
21. J.M. Intrieri et al., "An Annual Cycle of Arctic Surface Cloud Forcing at SHEBA," *Annals of Glaciology*, vol. 107, no. 10, 2002; doi:1029/2000JC000439.
22. Y. Chen et al., "Observed Relationships between Arctic Longwave Cloud Forcing and Cloud Parameters Using a Neural Network," *J. Climate*, vol. 19, no. 16, 2006, pp. 4087–4104.
23. I. Gorodetskaya et al., "The Influence of Cloud and Surface Properties on the Arctic Ocean Short Wave Radiation Budget in Coupled Models," to be published in *J. Climate*, 2007.
24. H.L. Stern, D.A. Rothrock, and R. Kwok, "Open Water Production in Arctic Sea Ice: Satellite Measurements and Model Parameterizations," *J. Geophysical Research*, vol. 100, Oct. 1995, pp. 20601–20612.
25. M.G. McPhee et al., "Upwelling of Arctic Pycnocline Associated with Shear Motion of Sea Ice," *Geophysical Research Letters*, vol. 32, May 2005; doi:10.1029/2004GL021819.
26. M. Steele and T. Boyd, "Retreat of the Cold Halocline Layer in the Arctic Ocean," *J. Geophysical Research*, vol. 103, May 1998, pp. 10419–10435.
27. G. Bjork et al., "Return of the Cold Halocline Layer to the Amundsen Basin of the Arctic Ocean: Implications for the Sea Ice Mass Balance," *Geophysical Research Letters*, vol. 29, June 2002; doi:10.1029/2001GL014157.
28. P. Schlosser et al., "Decrease of River Runoff in the Upper Waters of the Eurasian Basin, Arctic Ocean, between 1991 and 1996: Evidence from Delta 0-18 Data," *Geophysical Research Letters*, vol. 29, no. 9, 2002.
29. D.G. Martinson and R. Iannuzzi, "Antarctic Ocean-Ice Interactions: Implication from Ocean Bulk Property Distributions in the Weddell Gyre," *Antarctic Sea Ice: Physical Processes, Interactions, and Variability, Antarctic Research Series*, vol. 74, 1998, pp. 243–271.
30. D. Quadfasel et al., "Warming in the Arctic," *Nature*, vol. 350, no. 6317, 1991, p. 385.
31. M.J. Karcher et al., "Arctic Warming: Evolution and Spreading of the 1990s Warm Event in the Nordic Seas and the Arctic Ocean," *J. Geophysical Research*, vol. 108, June 2003; doi:10.1029/2001JC001265.
32. C. Bitz, J.C. Fyfe, and G.M. Flato, "Sea Ice Response to Wind Forcing from AMIP Models," *J. Climate*, vol. 15, no. 5, 2002, pp. 522–536.
33. G.L. Stephens et al., "The CloudSat Mission and the A-Train: A New Dimension of Space-Based Observations of Clouds and Precipitation," *Bulletin Am. Meteorological Soc.*, vol. 83, no. 12, 2002, pp. 1771–1790.
34. S. Kato and N.G. Loeb, "Top-of-Atmosphere Shortwave Broadband Observed Radiance and Estimated Irradiance over Polar Regions from Clouds and the Earth's Radiant Energy System (CERES) Instruments on Terra," *J. Geophysical Research*, vol. 110, Apr. 2005; doi: 10.1029/2004JD005308.
35. L.-B. Tremblay and L.A. Mysak, "Modeling Sea Ice as a Granular Material, Including the Dilatancy Effect," *J. Physical Oceanography*, vol. 27, no. 11, 1997, pp. 2342–2360.

***L. Bruno Tremblay** is an assistant professor at McGill University and an Adjunct Doherty Research Scientist at the Lamont-Doherty Earth Observatory of Columbia University. His research interests include Arctic climate and climate change, Arctic hydrology, sea-ice dynamics, and thermodynamic modeling. Tremblay has a PhD in atmospheric and oceanic sciences from McGill University. Contact him at bruno.tremblay@mcgill.ca.*

***Marika M. Holland** is a climate scientist at the US National Center for Atmospheric Research (NCAR). Her research interests include high-latitude climate variability with a particular focus on the role of sea ice in the climate system. Holland has a PhD in atmosphere and ocean sciences from the University of Colorado. Contact her at mholland@ucar.edu.*

***Irina V. Gorodetskaya** is a PhD candidate in the Department of Earth and Environmental Sciences at Columbia University. Her interests include clouds and ice in the Arctic , large-scale data set analysis, and process modeling. Contact her at irina@ldeo.columbia.edu.*

***Gavin A. Schmidt** is a climate scientist at the NASA Goddard Institute for Space Studies in New York, where he also works on climate-model development and evaluation. Schmidt has a PhD in applied mathematics from the University of London. Contact him at gschmidt@giss.nasa.gov.*