TheMatrixExponential

April 18, 2016

1 The Matrix Exponential

This notebook gives a brief introduction to the matrix exponential, an interesting function which can be found within many areas of applied mathematics. After defining the function we will introduce some applications and algorithms before giving links to currently available software and further information.

1.1 Definition

```
In [1]: %matplotlib inline
        import numpy as np
        import scipy as sp
        import scipy.linalg as la
        import matplotlib.pyplot as plt
```

The matrix exponential is a direct generalisation of the corresponding scalar function. Remember that the Taylor series expansion for $\exp(x)$ is

$$\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!}.$$

We can apply this to square matrices using matrix multiplcation:

$$\exp(A) = \sum_{k=0}^{\infty} \frac{A^k}{k!}.$$

Two of the most useful properties of the matrix exponential (see [11] for more) are that

$$\exp((a+b)A) = \exp(aA)\exp(bA),$$

and when two matrices A and B satisfy $AB = BA$,

$$\exp(A+B) = \exp(A)\exp(B).$$

A thorough treatment of the theory behind the matrix exponential (and matrix functions in general) can be found in [11].

1.2 Applications

The matrix exponential has a wide variety of applications in areas including cancer research [10] (which also uses the derivative of the exponential), nuclear burnup equations [12], and computer graphics [13]. We'll cover two here: solving systems of linear differential equations and finding the important nodes in a complex network.

1.2.1 Differential Equations

One major attraction of the matrix exponential is the fact that it gives the explicit solution to the differential equation

$$y'(t) = Ay(t), \qquad y(0) = y_0,$$

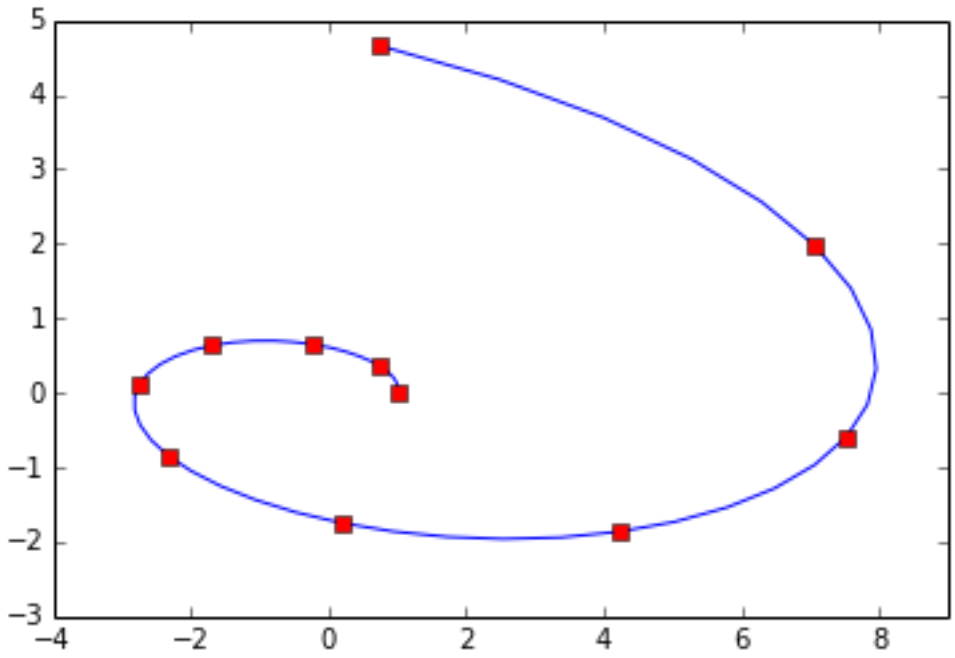
for $y, y_0 \in \mathbb{C}^n$ and $A \in \mathbb{C}^{n\times n}$. The solution is

$$y(t) = \exp(At)y_0.$$

Knowing this we can compute the solution $y(t)$ at any time point directly without using a time-stepping scheme such as the Euler method, for example. Let's do a simple example showing this for the differential equation

$$y'(t) = \begin{bmatrix} 1 & -20 \\ 3 & 4 \end{bmatrix} y(t), \qquad y(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad t \in [0, 1].$$

In [2]:
```
import scipy.integrate as sint

A = np.array([[1, -20],
              [3, 4]], dtype=np.double)

def yprime(y, t):
    return sp.dot(A, y)

t = np.linspace(0, 1, 51)
yzero = np.array([1, 0])

# Solve equation using odeint
y = sint.odeint(yprime, yzero, t) # Calls the Fortran library odeint
plt.plot(y[:, 0], y[:, 1]) # Blue line is y(t) for t in [0, 1]

plt.xlim([-4, 9])
plt.ylim([-3, 5])

# Solve for 10 t values using the exponential
tvals = np.linspace(0, 1, 11)

for tval in tvals:
    yval = sp.dot(la.expm(A*tval), yzero)
    plt.plot(yval[0], yval[1], 'rs') # Red squares are evaluated via the matrix exponential
```

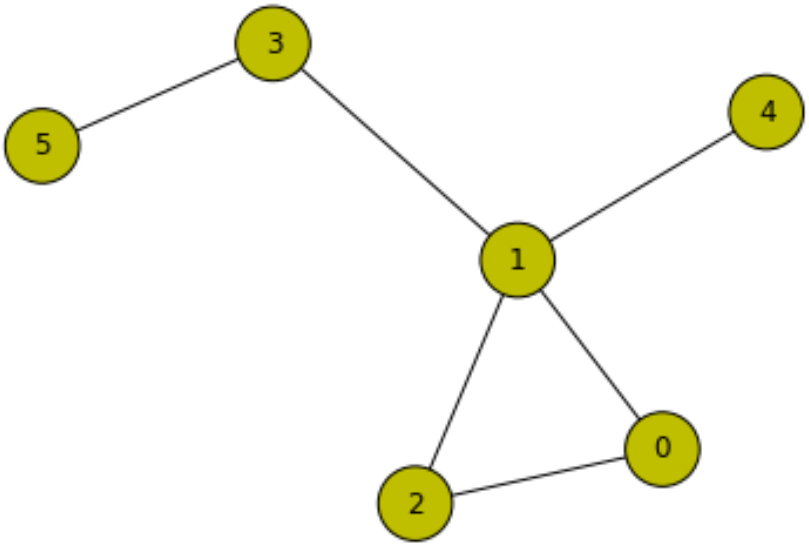


For our relatively simple ODE both methods perform well and, since the matrix exponential is more expensive to compute, we would probably opt to use scipy.integrate.odeint. For solving more complex differential equations the exponential forms the basis of a class of methods called exponential integrators [3]. Some more detail on these methods can be found in [7], for example. Note that in practice we would calculate $\exp(A)b$ without first computing $\exp(A)$ as this is cheaper. We could compute $\exp(A)b$ directly by using scipy.sparse.linalg.expm_multiply, which implements the recent algorithm from [3].

1.2.2 Important Nodes in a Network

Another recent application of the exponential is in finding the most influential nodes of a network. A network is a set of nodes connected by edges. We can represent a network by its adjacency matrix: we set $A_{ij} = 1$ if node i is connected to node j and $A_{ij} = 0$ otherwise. For example take the following network.

```
In [3]: import networkx as nx

        Adj = np.array([[0, 1, 1, 0, 0, 0],
                        [1, 0, 1, 1, 1, 0],
                        [1, 1, 0, 0, 0, 0],
                        [0, 1, 0, 0, 0, 1],
                        [0, 1, 0, 0, 0, 0],
                        [0, 0, 0, 1, 0, 0]])

        G = nx.from_numpy_matrix(Adj)
        nx.draw(G, node_color='y', node_size=1000)
```



One measure of the importance of each node is its centrality [4], [5], [6]. It is well known that the number of walks of length k from node i to node j is given by A^k_{ij}. We can add up all these contributions to obtain the centrality of node i as

$$c(i) = \alpha_1 A_{ii} + \alpha_2 A^2_{ii} + \alpha_3 A^3_{ii} + \cdots ,$$

where the coefficients α_k remain to be chosen. Typically we assume that shorter walks are more important than longer ones so $\alpha_k \geq \alpha_{k+1}$. There are a number of proposed formulae for α_k but taking $\alpha_k = \frac{1}{k!}$ we obtain $c(i) = \exp(A)$.

Looking at our network above we might expect node 1 to be the most important: it has the highest degree and links nodes 3, 4, and 5 to nodes 0 and 2. Let's work out the centrality of each node and rank them in order of importance.

```
In [4]: centralities = np.diag(la.expm(np.array(Adj, dtype=np.double)))
        nodeorder = np.argsort(centralities)[::-1]

        print np.array([nodeorder, centralities[nodeorder]])

        # Note: This is already built into networkx using the following command
        # print nx.communicability_centrality_exp(G)
```

```
[[ 1.          0.          2.          3.          4.          5.        ]
 [ 4.44723536  2.86427609  2.86427609  2.36018456  1.71615913  1.59432922]]
```

As we expected node 1 was the most important, followed by nodes 0 and 2 which have identical centralities since switching the two nodes wouldn't change the appearance of the graph.

1.3 Computing the Exponential

In this section we will give a basic algorithm to compute the matrix exponential. We will use the Taylor series (from the first section), which converges for every A but the convergence can be very slow when $\|A\|$

is large. To speed it up we can use the relationship

$$\exp\left(\frac{A}{p}\right)^p = \exp(A)$$

repeatedly with $p = 2$ to make $\|A/2^s\|$ small before applying the Taylor series and squaring the result s times. This is the basis of the scaling and squaring algorithm.

A number of potential methods for computing the exponential are described in detail within [8], [9].

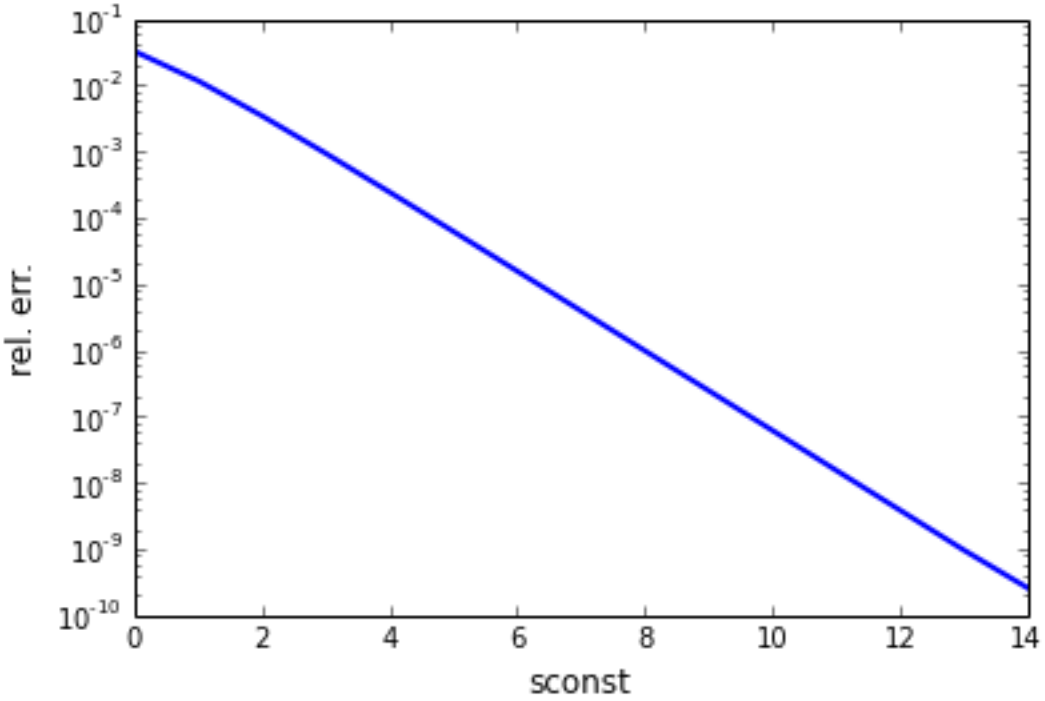
In [5]:
```
sconst = 1

def TaylorSS(A):
    taylordegree = 15 # Use order 15 Taylor approximation
    s = np.ceil(sp.log2(la.norm(A))) + sconst # Find s such that norm(A/2^s) is small.
    X = A/(2**s)
    eX = np.eye(np.shape(A)[0])
    for k in range(taylordegree): # Compute the Taylor series
        eX = eX + X/sp.misc.factorial(k+1)
        X = sp.dot(X, X)

    for k in range(np.int64(s)):
        eX = sp.dot(eX, eX) # Do the squaring phase of the algorithm

    return eX

#Let's test it against la.expm
A = np.random.randn(4, 4)
E1 = TaylorSS(A)
Eexact = la.expm(A)
print la.norm(E1 - Eexact)/la.norm(Eexact) # Relative error
```

```
0.0116677518269
```

Our algorithm worked but the relative error compared to scipy.linalg.expm is quite poor. If we increase sconst then $\|X\|$ will be much smaller and the Taylor series will be more accurate.

In [6]:
```
relerrs = np.zeros(15)
for sconst in range(15):
    E = TaylorSS(A)
    relerrs[sconst] = la.norm(E - Eexact)/la.norm(Eexact)

plt.plot(relerrs, color='blue', lw=2)
plt.yscale('log')
plt.xlabel('sconst', fontsize=12)
plt.ylabel('rel. err.', fontsize=12)
```

Out[6]:
```
<matplotlib.text.Text at 0x4441fd0>
```



This helps a lot but we are still not getting the same accuracy as scipy.linalg.expm.
The extra accuracy of scipy.linalg.expm is because it implements the algorithm from [2] which:
uses Pade approximants instead of Taylor series.
analyses the backward error of the function.
uses cost analysis to perform the least possible number of matrix multiplications.

1.4 Available Software

There is a lot of software available for computing the matrix exponential.

In Python: - scipy.linalg.expm - computes $\exp(A)$ using [2]. - scipy.sparse.linalg.expm_multiply - computes $\exp(A)B$ where $B \in \mathbb{C}^{n\times k}$ (useful for vectors B) using [3]. - scipy.linalg.expm_frechet - computes the Frechet derivative of the matrix exponential using [1]. - scipy.linalg.expm_cond (SciPy 0.14) - computes the condition number of the matrix exponential in the Frobenius norm using [11].

In MATLAB: - expm - built-in function evaluating $\exp(A)$ (currently using an older algorithm than SciPy). - expm_new - implements algorithms from [2]. - expmv - implements algorithms from [3]. - expm_frechet_pade - computes the derivative of $\exp(A)$ as explained in [11]. - expm_frechet_quad - computes the derivative of $\exp(A)$ as explained in [11]. - expm_cond - computes the condition number of $\exp(A)$ in the Frobenius norm as explained in [11]. - expokit - computes $\exp(A)$ for small dense A or $\exp(A)B$ for sparse A.

NAG Library (Fortran but callable in MATLAB, Python etc.): - F01(E/F)CF - Real / Complex matrix exponential using [2]. - F01(E/F)DF - computes the matrix exponential of a real symmetric / complex Hermitian matrix. - F01(G/H)AF - Real / Complex versions of $\exp(A)B$ using [3]. - F01(J/K)AF - estimates the condition number of the matrix exponential for real / complex input using [1].

1.5 Further Information

Some further information not directly referenced here can be found at: - Cleve's Corner - discusses the accuracy of computing $\exp(A)$. - Wikipedia - a more detailed introduction to the matrix exponential.

- Software Catalogue - A catalogue of available matrix function software (currently version 1.0) - Nick Higham's Blog - An short course on matrix functions (video lectures and written notes)

1.6 References

[1] - A. H. Al-Mohy and N. J. Higham, Computing the Frèchet Derivative of the Matrix Exponential, with an Application to Condition Number Estimation, SIAM J. Matrix Anal. Appl., 30:1639-1657, 2009.

[2] - A. H. Al-Mohy and N. J. Higham, A New Scaling and Squaring Algorithm for the Matrix Exponential, SIAM J. Matrix Anal. Appl., 31(3):970-989, 2009.

[3] - A. H. Al-Mohy and N. J. Higham, Computing the Action of the Matrix Exponential, with an Application to Exponential Integrators, SIAM J. Sci. Comput., 32:488-511, 2011.

[4] - Ernesto Estrada and N. Hatano, Communicability in Complex Networks, Phys. Rev. E., 77(3):036111, 2008.

[5] - Ernesto Estrada and D. J. Higham, Network Properties Revealed Through Matrix Functions, SIAM Rev. 52:696-714, 2010.

[6] - Ernesto Estrada, D. J. Higham, and N. Hatano, Communicability Betweeness in Complex Networks, Physica A: Statistical Mechanics and its Applications, 388:764-774, 2009.

[7] - Marlis Hochbruch and Alexander Osterman, Exponential Integrators, Acta Numerica, x:209-286, 2009.

[8] - C. B. Moler and C. F. Van-Loan, Nineteen Dubious Ways to Compute the Exponential of a Matrix, SIAM Rev., 20:801-836, 1978.

[9] - C. B. Moler and C. F. Van-Loan, Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later, SIAM Rev., 45:3-49, 2003.

[10] - B. Garcìa-Mora, C. Santamarìa, G. Rubio, and J. L. Pontones, Computing Survival Functions of the Sum of Two Independent Markov Processes. An Application to Bladder Carcinoma Treatment, Int. Journal of Computer Mathematics, 91(2):209-220, 2014.

[11] - N. J. Higham - Functions of Matrices: Theory and Computation, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008, ISBN: 978-0-898716-46-7, xx+425 pp.

[12] - Maria Pusa and Jaakko Leppănen Computing the Matrix Exponential in Burnup Calculations, Nuclear Science and Engineering, 164(2):140-150, 2010.

[13] - Jarek Rossignac and Àlvar Vinucua, Steady Affine Motions and Morphs, ACM Trans. Graph., 30(5):Article 116, 16 pages, 2011.