Example: Solving differential equations numerically

§1.4

Problem:

Find a function f on $[0,1]$ with

$$f''(t)=\sqrt{t}$$
$$f(0)=f(1)=0$$

Answer:

$$f''(t)=\sqrt{t} \quad \Rightarrow f'(t) = c+\int_0^t f''(s)\,ds$$
$$= \tfrac{2}{3}t^{3/2}+c$$
$$\Rightarrow f(t) = \tfrac{4}{15}t^{5/2}+ct+d$$

for some constants c, d

$$f(0)=0 \Rightarrow d=0$$
$$f(1)=\tfrac{4}{15}+c=0 \Rightarrow c=-\tfrac{4}{15} \qquad \Rightarrow \boxed{f(t)=\tfrac{4}{15}\left(t^{5/2}-t\right)}$$

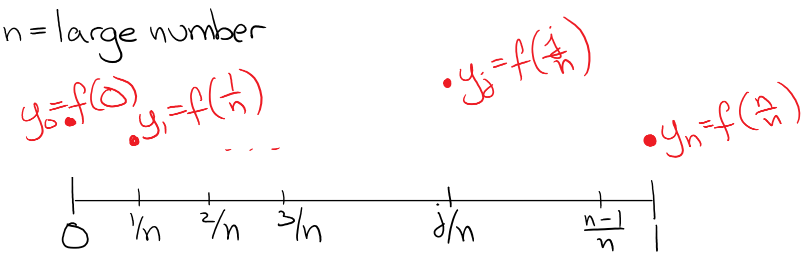
This worked because we have closed-form expressions for $\int\sqrt{s}\,ds$ and $\int s^{3/2}ds$. But frequently there won't be closed-form integrals.

e.g., $\operatorname{erf}(x) := \frac{2}{\sqrt{\pi}}\int_0^x e^{-s^2}ds$. See also https://math.stackexchange.com/questions/155/how-can-you-prove-that-a-function-has-no-closed-form-integral

Then what?

We can solve a differential equation numerically (approximately) by discretizing the domain, $[0,1]$.

n = large number



$n+1$ variables $y_0, \ldots, y_n$

Equations:

$$f''(t)=\sqrt{t} \ ?$$
$$f(0)=f(1)=0 \rightarrow \boxed{y_0=y_n=0}$$

Recall

$$f'(t)=\frac{d}{dt}f(t) := \lim_{\delta\to 0}\frac{f(t+\delta)-f(t)}{\delta}$$
$$\approx \frac{f(t+\frac{1}{n})-f(t)}{1/n}$$

$$f''(t)=\frac{d}{dt}f'(t) \approx \frac{f'(t)-f'(t-1/n)}{1/n}$$
$$\approx n^2\left[\left(f(t+\tfrac{1}{n})-f(t)\right)-\left(f(t)-f(t-\tfrac{1}{n})\right)\right]$$
$$= n^2\left[f(t+\tfrac{1}{n})-2f(t)+f(t-\tfrac{1}{n})\right]$$

$$\Rightarrow \text{at } t=\tfrac{j}{n}, \quad f''(t) = \sqrt{t}$$
$$n^2\left(y_{j-1}-2y_j+y_{j+1}\right) \approx \sqrt{\tfrac{j}{n}}$$

$$n^2(y_{j-1} - 2y_j + y_{j+1}) \approx \sqrt{\tfrac{j}{n}}$$

$\Rightarrow$ for $j = 1, \dots, n-1$, $\boxed{y_{j-1} - 2y_j + y_{j+1} = \frac{\sqrt{j}}{n^{5/2}}}$

Overall, $n+1$ sparse! equations for $n+1$ variables.

Sparse matrices in Mathematica, Matlab, Python

Goal: Solve numerically the differential equation

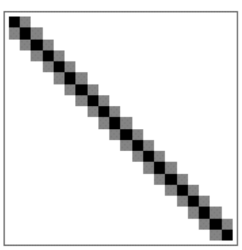
$$\begin{cases} f''(t) = \sqrt{t} \\ f(0) = f(1) = 0 \end{cases}$$

by discretizing the interval $[0, 1]$.

Mathematica

$$x_{j+1} - 2x_j + x_{j-1} == \frac{1}{n^2}\sqrt{\frac{j}{n}} \qquad x_j = f\left(\tfrac{j}{n}\right)$$

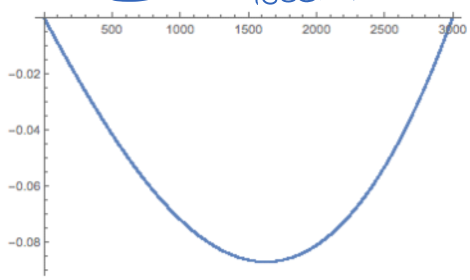
```
n = 3000;

b = Table[1/n^2 Sqrt[j/n], {j, 1, n - 1}] // N;

A = SparseArray[Join[
     Table[{j, j} -> -2, {j, 1, n - 1}],
     Table[{j, j + 1} -> 1, {j, 1, n - 2}],
     Table[{j + 1, j} -> 1, {j, 1, n - 2}]
    ]];

ArrayPlot[A[[ ;; 20, ;; 20]]]
```



```
Timing[
 x = LinearSolve[A, b];
]

{0.002488, Null}

Afull = Normal[A];
Timing[
 x = LinearSolve[Afull, b];
]

ListPlot[x]

{2.21352, Null}
```

converts A to a dense matrix

sparse matrix solving 1000x faster!!

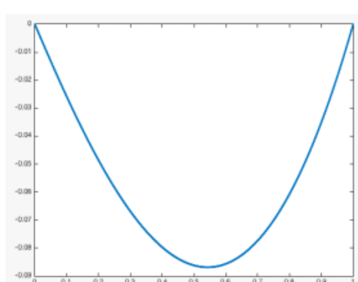


Matlab commands

```
>> n = 3000;

b = 1/n^2.5 * sqrt(1:n-1)';

A = spdiags(ones(n-2,1),-1,n-1,n-1);
A = A + A' - 2*spdiags(ones(n-1,1),0,n-1,n-1);
full(A(1:9,1:9))

size(A)

ans =

    -2     1     0     0     0     0     0     0     0
     1    -2     1     0     0     0     0     0     0
     0     1    -2     1     0     0     0     0     0
     0     0     1    -2     1     0     0     0     0
     0     0     0     1    -2     1     0     0     0
     0     0     0     0     1    -2     1     0     0
     0     0     0     0     0     1    -2     1     0
     0     0     0     0     0     0     1    -2     1
     0     0     0     0     0     0     0     1    -2


ans =

        2999        2999
```

```
>> tic
x = A \ b;
toc
Elapsed time is 0.010280 seconds.
>> Afull = full(A);
tic
x = Afull \ b;
toc
Elapsed time is 0.999726 seconds.

>> plot(1/n*(1:n-1),x,'.');
axis([0 1 -.1 0]);
```

converts A to a dense matrix



Note: $A = \text{sparse}([i_1, i_2, i_3], [j_1, j_2, j_3], [s_1, s_2, s_3])$ creates a sparse matrix with $a_{i_1 j_1} = s_1$, $a_{i_2 j_2} = s_2$, $a_{i_3 j_3} = s_3$.

```
n = 100;
A1 = sparse(1:n-1, 1:n-1, -2);
```

range $[1, 2, 3, \dots, n-1]$ — rows — columns

creates $(n-1)\times(n-1)$ sparse matrix with -2 on diagonal

```
n = 100;
A1 = sparse(1:n-1, 1:n-1, -2);
A2 = sparse(2:n-1, 1:n-2, 1, n-1,n-1);
A3 = sparse(1:n-2, 2:n-1, 1, n-1,n-1);
A = A1+A2+ A3;
A
spy(A, 'o', 1);
```

range [1, 2, 3, ..., n-1]; rows; columns

creates $(n-1)\times(n-1)$ sparse matrix with -2 on diagonal

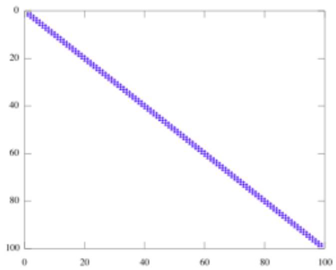
A2 has +1's just below the diagonal

A3 has +1's just above the diagonal

make it an $(n-1)\times(n-1)$ matrix instead of $(n-2)\times(n-1)$

tridiagonal

"spy" command to visualize the matrix



Equivalent code to define A:

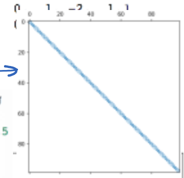
```
e = ones(n, 1);
A = spdiags([e, -2*e, e], -1:1, n, n);
```

Python

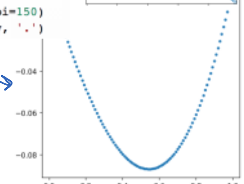
```python
import numpy as np
from scipy import sparse

n = 100
e = np.ones(n)
A = sparse.spdiags([e, -2*e, e], [-1,0,1], n-1, n-1, format='csr')
print(A[:5,:5].toarray())
```

```
[[-2.  1.  0.  0.  0.]
 [ 1. -2.  1.  0.  0.]
 [ 0.  1. -2.  1.  0.]
 [ 0.  0.  1. -2.  1.]
 [ 0.  0.
```

```python
import matplotlib.pyplot as plt
plt.figure(figsize=(5,5), dpi=150)
plt.spy(A, markersize=1)
```



```python
from scipy.sparse.linalg import bicg

b = (np.arange(1,n) ** .5) / n ** 2.5
y, exitCode = bicg(A, b)

plt.figure(figsize=(5,5), dpi=150)
plt.plot(np.arange(1,n)/n, y, '.')
```



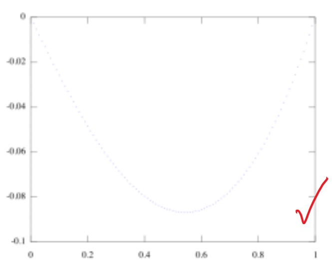
```
b = n^(-2.5) * (1:n-1)' .^ (1/2);
y = A\b;
plot((1:n-1)/100, y, '.');
```

transpose to get column vector

solve for y!

.^ gives element-wise exponentiation

x-coordinates; y-coords; plot marker



✓

Extension: What if we had a differential equation in two variables, x and y?
(On, eg., the square $[0,1]\times[0,1]$.)
What would the matrix A look like?

Convergence speed:

How good are the approximations

$$f'(t) \approx n\left(f(t+\tfrac{1}{n}) - f(t)\right)$$

$$f''(t) \approx n^2\left[f(t+\tfrac{1}{n}) - 2f(t) + f(t-\tfrac{1}{n})\right] \quad ?$$

Recall: Taylor series https://en.wikipedia.org/wiki/Taylor%27s_theorem

$$f(x+\delta) = f(x) + \delta\cdot f'(x) + \tfrac{1}{2}\delta^2\cdot f''(x) + o(\delta^2)$$

$$\Rightarrow\ n\left(f(t+\tfrac{1}{n}) - f(t)\right) = n\left[\tfrac{1}{n}f'(t) + \tfrac{1}{2}\tfrac{1}{n^2}f''(t) + o(\tfrac{1}{n^3})\right]$$

$$= f'(t) + O\left(\tfrac{1}{n}\right)$$

approximation error drops like $\frac{1}{n}$

$$n^2\left[-2f(t) + f(t-\tfrac{1}{n}) + f(t+\tfrac{1}{n})\right]$$

$$= n^2\left[-\cancel{\tfrac{1}{n}f'(t)} + \tfrac{1}{2}\tfrac{1}{n^2}f''(t) + O(\tfrac{1}{n^3}) + \cancel{\tfrac{1}{n}f'(t)} + \tfrac{1}{2}\tfrac{1}{n^2}f''(t) + O(\tfrac{1}{n^3})\right]$$

$$n^2\left[-2f(t)+f\left(t-\tfrac{1}{n}\right)+f\left(t+\tfrac{1}{n}\right)\right]$$

$$= n^2\left[\cancel{-\tfrac{1}{n}f'(t)}+\tfrac{1}{2}\tfrac{1}{n^2}f''(t)+O\left(\tfrac{1}{n^3}\right)+\cancel{\tfrac{1}{n}f'(t)}+\tfrac{1}{2}\tfrac{1}{n^2}f''(t)+O\left(\tfrac{1}{n^3}\right)\right]$$

$$= f''(t)+n^2\cdot O\left(\tfrac{1}{n^3}\right)$$

$$= f''(t)+O\left(\tfrac{1}{n}\right)$$

approximation error drops like $\frac{1}{n}$

Remark: Consider

$$\frac{n}{2}\left(f\left(t+\tfrac{1}{n}\right)-f\left(t-\tfrac{1}{n}\right)\right)=\frac{n}{2}\left(\begin{array}{l}\underbrace{f(t)}_{\text{these cancel}}+\tfrac{1}{n}f'(t)+\underbrace{\tfrac{1}{2}\tfrac{1}{n^2}f''(t)}_{\text{these cancel}}+O\left(\tfrac{1}{n^3}\right)\\ -\left(f(t)-\tfrac{1}{n}f'(t)+\tfrac{1}{2}\tfrac{1}{n^2}f''(t)+O\left(\tfrac{1}{n^3}\right)\right)\end{array}\right)$$

$$=\cancel{\tfrac{n}{2}}\cdot\cancel{\tfrac{2}{n}}f'(t)+n\cdot O\left(\tfrac{1}{n^3}\right)$$

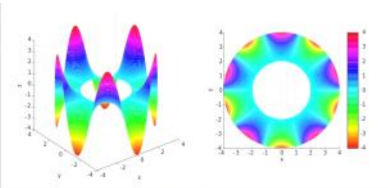
$$=f'(t)+O\left(\tfrac{1}{n^2}\right)$$

approximation error drops like $\frac{1}{n^2}$!

This is a better approximation for the first derivative.

2-dimensional example

This example is from https://en.wikipedia.org/wiki/Laplace%27s_equation



Laplace's equation on an annulus (inner radius $r = 2$ and outer radius $R = 4$) with Dirichlet boundary conditions $u(r=2) = 0$ and $u(R=4) = 4\sin(5\theta)$

In radial coordinates (r, θ),

$$\nabla^2 f = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial f}{\partial r}\right)+\frac{1}{r^2}\frac{\partial^2 f}{\partial \theta^2}=\frac{\partial^2 f}{\partial r^2}+\frac{1}{r}\frac{\partial f}{\partial r}+\frac{1}{r^2}\frac{\partial^2 f}{\partial \theta^2}$$

```python
import numpy as np
from scipy import sparse

innerRadius = 2
outerRadius = 4

nr, ntheta = 300, 500
ntotal = nr * ntheta

radii = innerRadius + (outerRadius - innerRadius) / (nr - 1) * np.arange(nr)
OneOverR = sparse.diags(np.concatenate([np.ones(ntheta) / r for r in radii]))

D2Dtheta2 = sparse.lil_matrix((ntotal, ntotal))
for ir in range(1, nr-1):  # only for the interior points
  for itheta in range(ntheta):
    i = ir * ntheta + itheta
    iright = ir * ntheta + ((itheta+1)%ntheta)  # periodic boundary conditions
    ileft  = ir * ntheta + ((itheta-1)%ntheta)
    D2Dtheta2[i,i] = -2
    D2Dtheta2[i,ileft], D2Dtheta2[i,iright] = 1, 1
D2Dtheta2 = D2Dtheta2.tocsr()
D2Dtheta2 *= (ntheta / (2 * np.pi)) ** 2

DDr = sparse.lil_matrix((ntotal, ntotal))
for ir in range(1, nr-1):  # only for the interior points
  for itheta in range(ntheta):
    i = ir * ntheta + itheta
    iout = i + ntheta
    DDr[i,iout], DDr[i,i] = 1, -1
DDr = DDr.tocsr()
DDr *= (nr - 1) / (outerRadius - innerRadius)

D2Dr2 = sparse.lil_matrix((ntotal, ntotal))
for ir in range(1, nr-1):  # only for the interior points
  for itheta in range(ntheta):
    i = ir * ntheta + itheta
    iin, iout = i - ntheta, i + ntheta
    D2Dr2[i,i] = -2
    D2Dr2[i,iin], D2Dr2[i,iout] = 1, 1
D2Dr2 = D2Dr2.tocsr()
D2Dr2 *= ((nr - 1) / (outerRadius - innerRadius)) ** 2
```

```python
boundaries = sparse.lil_matrix((ntotal, ntotal))
for ir in (0,nr-1):
  for itheta in range(ntheta):
    i = ir * ntheta + itheta
    boundaries[i,i] = 1
boundaries = boundaries.tocsr()
# note that boundaries is the identity on the first and last itheta rows
# the other matrices are 0 in these rows
```

```python
A = D2Dr2 + OneOverR * DDr + OneOverR * OneOverR * D2Dtheta2 + boundaries
b = np.concatenate(
    (np.zeros(ntheta),          # inner boundary
     np.zeros((nr-2)*ntheta),  # interior
     [4 * np.sin(5 * (2 * np.pi / ntheta) * i) for i in range(ntheta)]) # outer
    )
```

```python
import matplotlib.pyplot as plt
plt.figure(figsize=(5,5), dpi=150)
plt.spy(A, markersize=1)
```

```python
from scipy.sparse.linalg import spsolve, bicg, bicgstab, lsmr
#x, exitCode = bicg(A, b) # bicg isn't working well?
x = spsolve(A, b)
#x = lsmr(A, b)[0]  # iterative least-squares solver

np.allclose(A.dot(x), b)
```

```
True
```

```python
# now we plot the data in 3d

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(5,5), dpi=150)
ax = fig.add_subplot(111, projection='3d')

radii = innerRadius + (outerRadius - innerRadius) / (nr - 1) * np.arange(nr)
xvals = np.concatenate([[r * np.cos((2 * np.pi / ntheta) * i) for i in range(ntheta)]
                        for r in radii])
yvals = np.concatenate([[r * np.sin((2 * np.pi / ntheta) * i) for i in range(ntheta)]
                        for r in radii])
ax.plot_trisurf(xvals, yvals, x, linewidth=0, antialiased=True, cmap=plt.cm.Spectral)
plt.show()
```