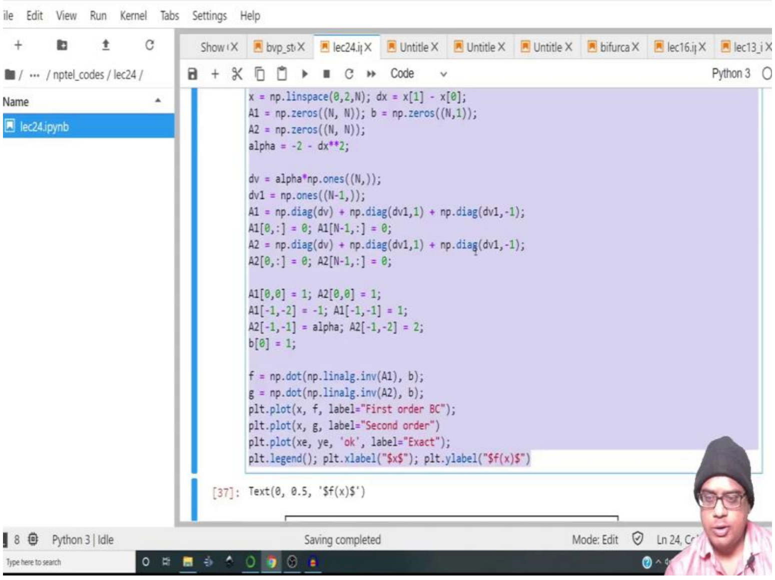
Tools in Scientific Computing

Prof. Aditya Bandopadhyay
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture - 25
Boundary Value Problems - p2

(Refer Slide Time: 00:26)

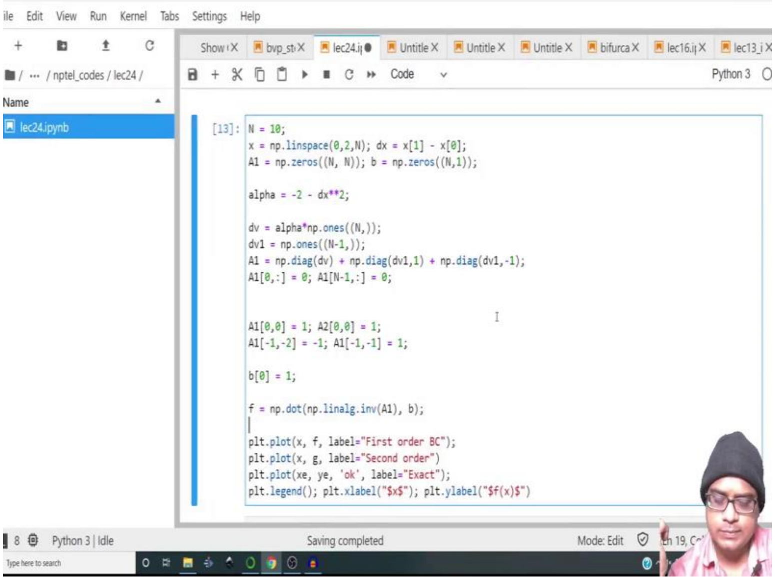

ile Edit View Run Kernel Tabs Settings Help
/ ··· / nptel_codes / lec24 /
Name
lec24.ipynb
Show (bvp_st lec24.i Untitle Untitle Untitle bifurca lec16.i lec13_i
Code
Python 3
8 Python 3 | Idle
Saving completed
Mode: Edit
Ln 24, C
Type here to search

```
x = np.linspace(0,2,N); dx = x[1] - x[0];
A1 = np.zeros((N, N)); b = np.zeros((N,1));
A2 = np.zeros((N, N));
alpha = -2 - dx**2;

dv = alpha*np.ones((N,));
dv1 = np.ones((N-1,));
A1 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
A1[0,:] = 0; A1[N-1,:] = 0;
A2 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
A2[0,:] = 0; A2[N-1,:] = 0;

A1[0,0] = 1; A2[0,0] = 1;
A1[-1,-2] = -1; A1[-1,-1] = 1;
A2[-1,-1] = alpha; A2[-1,-2] = 2;
b[0] = 1;

f = np.dot(np.linalg.inv(A1), b);
g = np.dot(np.linalg.inv(A2), b);
plt.plot(x, f, label="First order BC");
plt.plot(x, g, label="Second order")
plt.plot(xe, ye, 'ok', label="Exact");
plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")
```

```
[37]: Text(0, 0.5, '$f(x)$')
```

Hi everyone. Welcome to this lecture. So, in the last class, we had looked at how to solve an ODE and I had mentioned that we will look at some other means of finding out the inverse of the matrix.

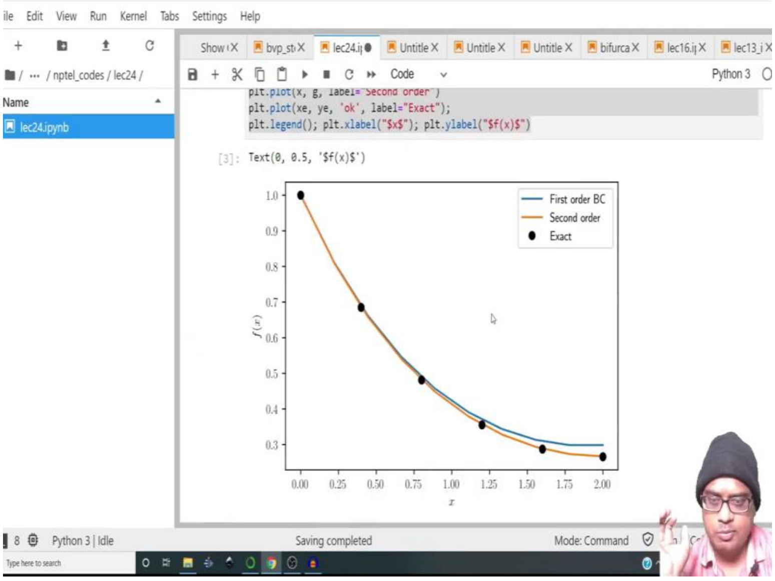
(Refer Slide Time: 00:54)



```
[13]: N = 10;
x = np.linspace(0,2,N); dx = x[1] - x[0];
A1 = np.zeros((N, N)); b = np.zeros((N,1));

alpha = -2 - dx**2;

dv = alpha*np.ones((N,));
dv1 = np.ones((N-1,));
A1 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
A1[0,:] = 0; A1[N-1,:] = 0;


A1[0,0] = 1; A2[0,0] = 1;
A1[-1,-2] = -1; A1[-1,-1] = 1;

b[0] = 1;

f = np.dot(np.linalg.inv(A1), b);

plt.plot(x, f, label="First order BC");
plt.plot(x, g, label="Second order")
plt.plot(xe, ye, 'ok', label="Exact");
plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")
```

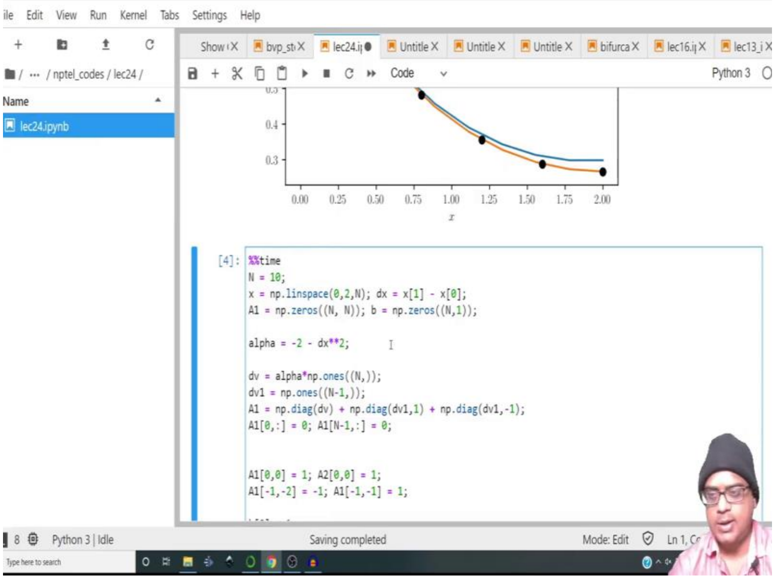
So, let me take this particular code. Let me yeah let me get rid of the cell. I do not need this cell as well. So, let me get rid of all the A 2s. We do not need the A 2 bit alright. So, let me quickly run this entire sheet again. So, what we will try to do first is to time this particular cell.

(Refer Slide Time: 01:29)



```
plt.plot(x, g, label="Second order")
plt.plot(xe, ye, 'ok', label="Exact");
plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")

[3]: Text(0, 0.5, '$f(x)$')
```

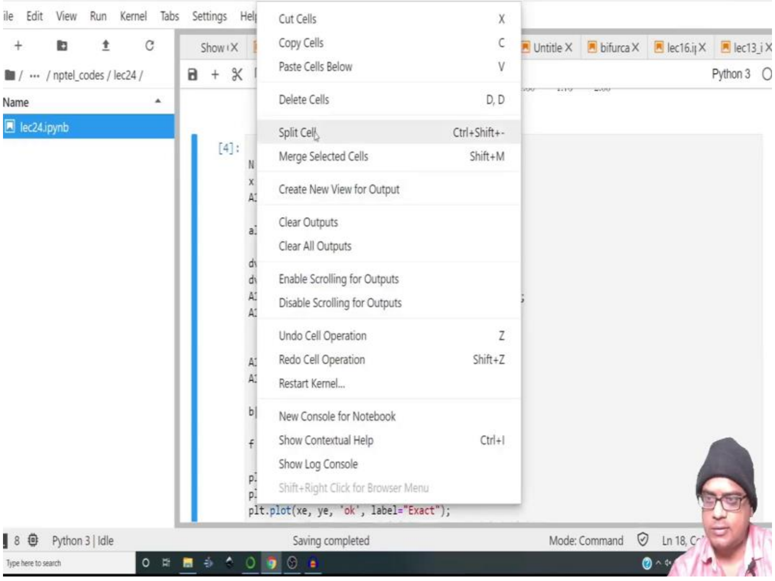
(Refer Slide Time: 01:37)



```
[4]: %%time
     N = 10;
     x = np.linspace(0,2,N); dx = x[1] - x[0];
     A1 = np.zeros((N, N)); b = np.zeros((N,1));

     alpha = -2 - dx**2;

     dv = alpha*np.ones((N,));
     dv1 = np.ones((N-1,));
     A1 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
     A1[0,:] = 0; A1[N-1,:] = 0;


     A1[0,0] = 1; A2[0,0] = 1;
     A1[-1,-2] = -1; A1[-1,-1] = 1;
```

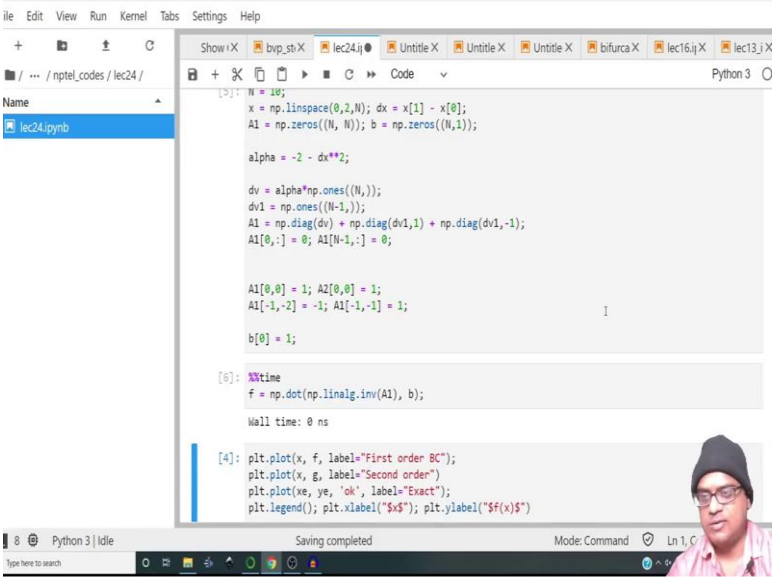
So, what does timing a particular cell mean? It means that it will tell us how much time it took for that particular cell to run. In particular, we are more concerned with this particular ,nd. So, let me not write time over here; let me split this cell.

(Refer Slide Time: 01:57)



```
plt.plot(xe, ye, 'ok', label="Exact");
```

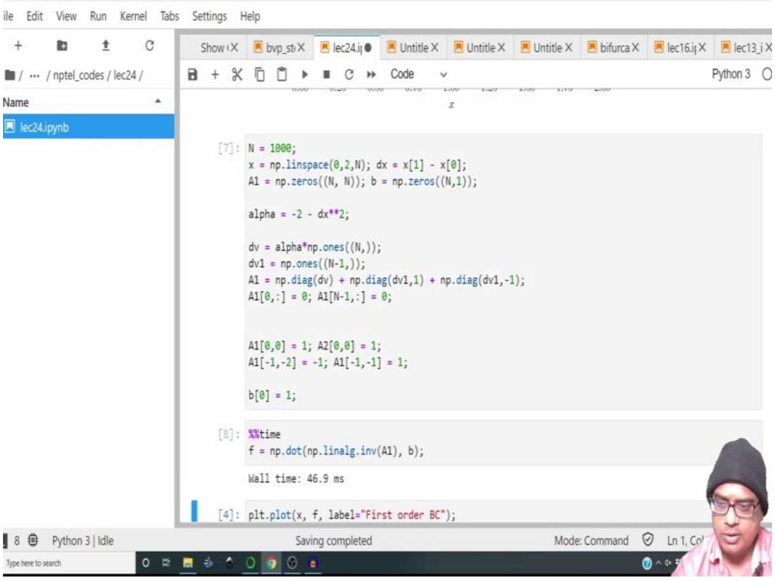
(Refer Slide Time: 01:59)


File Edit View Run Kernel Tabs Settings Help
/ ··· / nptel_codes / lec24 /
Name
lec24.ipynb
Show (| bvp_st | lec24.i | Untitle | Untitle | Untitle | bifurca | lec16.i | lec13_i
Code
Python 3
8 Python 3 | Idle
Saving completed
Mode: Command
Ln 1, C

```
[5]: N = 10;
     x = np.linspace(0,2,N); dx = x[1] - x[0];
     A1 = np.zeros((N, N)); b = np.zeros((N,1));

     alpha = -2 - dx**2;

     dv = alpha*np.ones((N,));
     dv1 = np.ones((N-1,));
     A1 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
     A1[0,:] = 0; A1[N-1,:] = 0;


     A1[0,0] = 1; A2[0,0] = 1;
     A1[-1,-2] = -1; A1[-1,-1] = 1;

     b[0] = 1;

[6]: %%time
     f = np.dot(np.linalg.inv(A1), b);

     Wall time: 0 ns

[4]: plt.plot(x, f, label="First order BC");
     plt.plot(x, g, label="Second order")
     plt.plot(xe, ye, 'ok', label="Exact");
     plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")
```

So, let me split this over here and let me split this over here. So, let me run this cell first. Now, before running this, let me do percentage-percentage time. So, it will give me how much time it took for that particular cell to evaluate. It is 0 nanosecond, because I have taken a very low grid count.

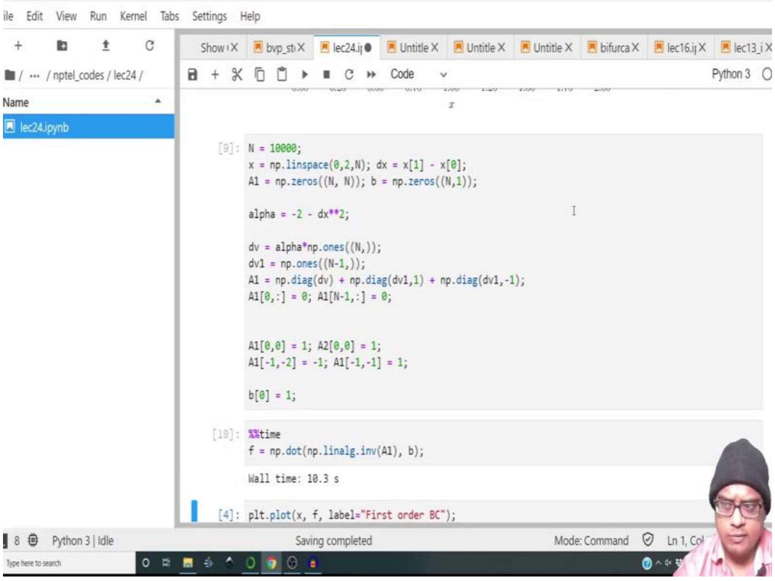
(Refer Slide Time: 02:22)


File Edit View Run Kernel Tabs Settings Help
/ ··· / nptel_codes / lec24 /
Name
lec24.ipynb
Show (| bvp_st | lec24.i | Untitle | Untitle | Untitle | bifurca | lec16.i | lec13_i
Code
Python 3
x
8 Python 3 | Idle
Saving completed
Mode: Command
Ln 1, Co

```
[7]: N = 1000;
     x = np.linspace(0,2,N); dx = x[1] - x[0];
     A1 = np.zeros((N, N)); b = np.zeros((N,1));

     alpha = -2 - dx**2;

     dv = alpha*np.ones((N,));
     dv1 = np.ones((N-1,));
     A1 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
     A1[0,:] = 0; A1[N-1,:] = 0;


     A1[0,0] = 1; A2[0,0] = 1;
     A1[-1,-2] = -1; A1[-1,-1] = 1;

     b[0] = 1;

[8]: %%time
     f = np.dot(np.linalg.inv(A1), b);

     Wall time: 46.9 ms

[4]: plt.plot(x, f, label="First order BC");
```

So, let me take 1000 ok; let me run it. So, it took 46.9 milliseconds to perform the inversion.

(Refer Slide Time: 02:34)



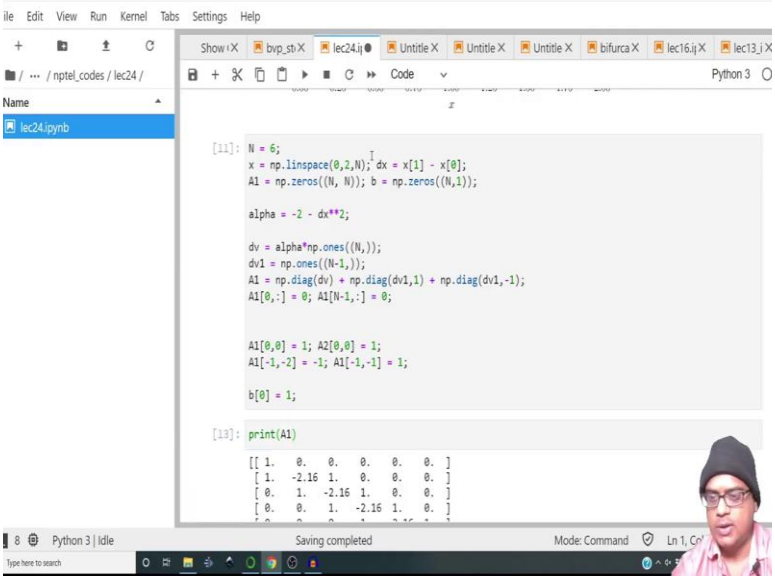
```
[9]: N = 10000;
     x = np.linspace(0,2,N); dx = x[1] - x[0];
     A1 = np.zeros((N, N)); b = np.zeros((N,1));

     alpha = -2 - dx**2;

     dv = alpha*np.ones((N,));
     dv1 = np.ones((N-1,));
     A1 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
     A1[0,:] = 0; A1[N-1,:] = 0;


     A1[0,0] = 1; A2[0,0] = 1;
     A1[-1,-2] = -1; A1[-1,-1] = 1;

     b[0] = 1;
```

```
[10]: %%time
      f = np.dot(np.linalg.inv(A1), b);

      Wall time: 10.3 s
```

```
[4]: plt.plot(x, f, label="First order BC");
```

And as time, as the number of grid points increases, so I have made it from 1000 to 10000. So, for 1000 grid points, it took 46.9, let us say 50 milliseconds; let us see how much time it takes for 10000 grid points. This may take a while. If it is linearly scaling, it will take almost 5 seconds or rather point. So, it was 50 milliseconds. So, yeah. So, it took 10 seconds. So, it is not at all scalable. So, then why is it not scalable ok?

(Refer Slide Time: 03:15)

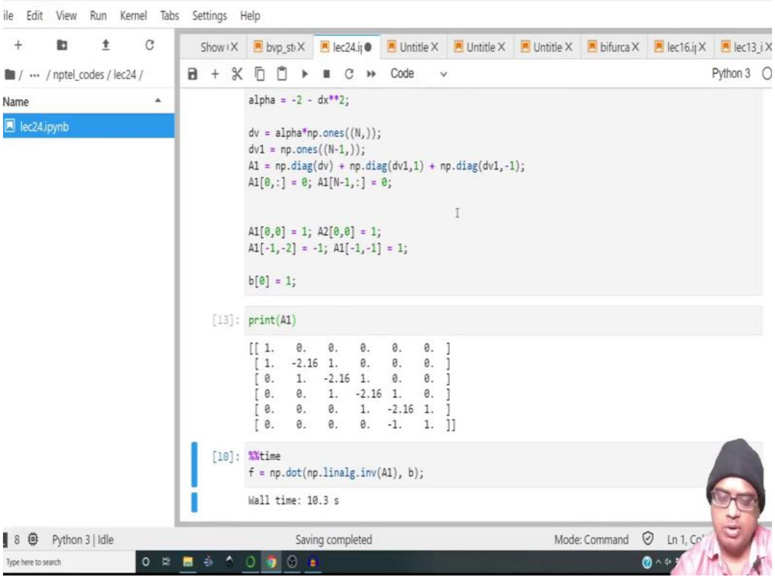


```
[11]: N = 6;
      x = np.linspace(0,2,N); dx = x[1] - x[0];
      A1 = np.zeros((N, N)); b = np.zeros((N,1));

      alpha = -2 - dx**2;

      dv = alpha*np.ones((N,));
      dv1 = np.ones((N-1,));
      A1 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
      A1[0,:] = 0; A1[N-1,:] = 0;


      A1[0,0] = 1; A2[0,0] = 1;
      A1[-1,-2] = -1; A1[-1,-1] = 1;

      b[0] = 1;
```

```
[13]: print(A1)

      [[ 1.    0.    0.    0.    0.    0.  ]
       [ 1.   -2.16  1.    0.    0.    0.  ]
       [ 0.    1.   -2.16  1.    0.    0.  ]
       [ 0.    0.    1.   -2.16  1.    0.  ]
```

So, before going into the scalability and all that thing, let me make N to be 6, just to show you how again I mean just to show how A looks like. So, let me just print A over here; sorry, this should be A 1.

(Refer Slide Time: 03:29)



ile Edit View Run Kernel Tabs Settings Help
/ ··· / nptel_codes / lec24 /
Name
lec24.ipynb
Show (bvp_st lec24.i Untitle Untitle Untitle bifurca lec16.i lec13_i
Code
Python 3
8 Python 3 | Idle
Saving completed
Mode: Command
Ln 1, Co
Type here to search

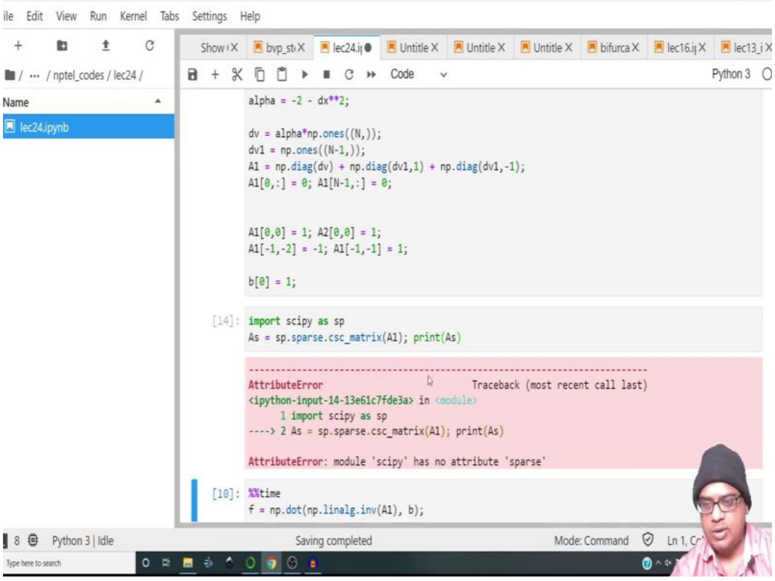
```
alpha = -2 - dx**2;

dv = alpha*np.ones((N,));
dv1 = np.ones((N-1,));
A1 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
A1[0,:] = 0; A1[N-1,:] = 0;


A1[0,0] = 1; A2[0,0] = 1;
A1[-1,-2] = -1; A1[-1,-1] = 1;

b[0] = 1;
```

```
[13]: print(A1)

[[ 1.    0.    0.    0.    0.    0.  ]
 [ 1.   -2.16  1.    0.    0.    0.  ]
 [ 0.    1.   -2.16  1.    0.    0.  ]
 [ 0.    0.    1.   -2.16  1.    0.  ]
 [ 0.    0.    0.    1.   -2.16  1.  ]
 [ 0.    0.    0.    0.   -1.    1.  ]]
```

```
[10]: %%time
f = np.dot(np.linalg.inv(A1), b);

Wall time: 10.3 s
```

So, A 1 is said to be a sparse matrix; meaning, most of the entries in A are zeros and it is a diagonal sparse matrix meaning, most of the elements of A are concentrated around the diagonal. So, now, there are a host of different algorithms which exist for dealing with sparse matrices. And the fact of the matter is you do not really need that much of an memory overhead or algorithmic overhead. Because you know you are just keeping a count of elements which are non-zero ok. So, A is like this.

(Refer Slide Time: 04:13)



```
alpha = -2 - dx**2;

dv = alpha*np.ones((N,));
dv1 = np.ones((N-1,));
A1 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
A1[0,:] = 0; A1[N-1,:] = 0;


A1[0,0] = 1; A2[0,0] = 1;
A1[-1,-2] = -1; A1[-1,-1] = 1;

b[0] = 1;
```

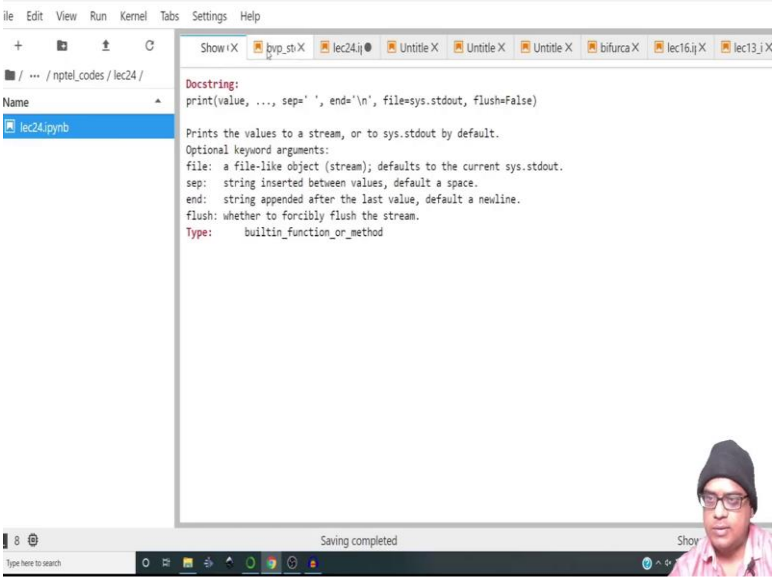
```
[14]: import scipy as sp
      As = sp.sparse.csc_matrix(A1); print(As)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-14-13e61c7fde3a> in <module>
      1 import scipy as sp
----> 2 As = sp.sparse.csc_matrix(A1); print(As)

AttributeError: module 'scipy' has no attribute 'sparse'
```

```
[10]: %%time
      f = np.dot(np.linalg.inv(A1), b);
```

So, let me do this. So, let me import scipy dot ok. So, let me import scipy as sp alright. So, sp.sparse. So, this is the name of the ,nd and let me dot csc matrix. So, it is compressed yeah. So, what is the full form?

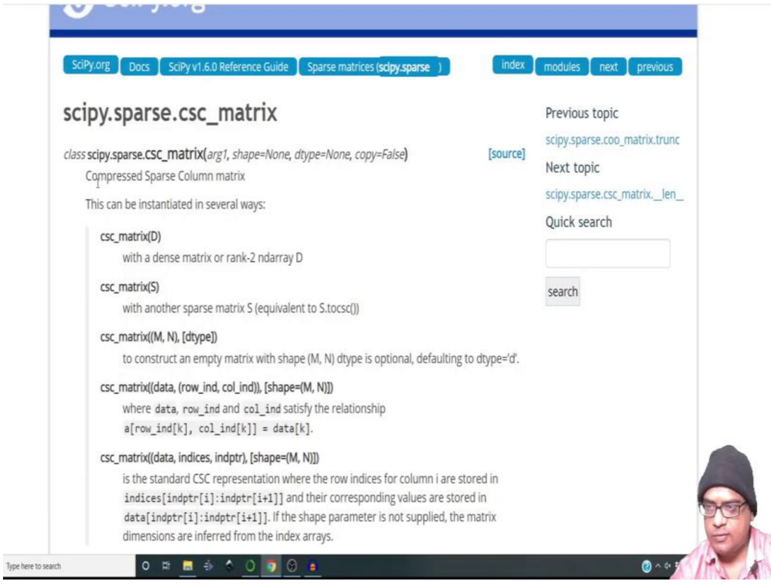
(Refer Slide Time: 04:46)



```
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

So, it is again, we can simply have a look at the yeah csc stands for compressed sparse column matrix.

(Refer Slide Time: 04:53)



scipy.sparse.csc_matrix

class scipy.sparse.csc_matrix(*arg1*, *shape=None*, *dtype=None*, *copy=False*) [source]

Compressed Sparse Column matrix

This can be instantiated in several ways:

csc_matrix(D)

with a dense matrix or rank-2 ndarray D

csc_matrix(S)

with another sparse matrix S (equivalent to S.tocsc())

csc_matrix((M, N), [dtype])

to construct an empty matrix with shape (M, N) dtype is optional, defaulting to dtype='d'.

csc_matrix((data, (row_ind, col_ind)), [shape=(M, N)])

where `data`, `row_ind` and `col_ind` satisfy the relationship `a[row_ind[k], col_ind[k]] = data[k]`.

csc_matrix((data, indices, indptr), [shape=(M, N)])

is the standard CSC representation where the row indices for column i are stored in `indices[indptr[i]:indptr[i+1]]` and their corresponding values are stored in `data[indptr[i]:indptr[i+1]]`. If the shape parameter is not supplied, the matrix dimensions are inferred from the index arrays.

Previous topic

scipy.sparse.coo_matrix.trunc

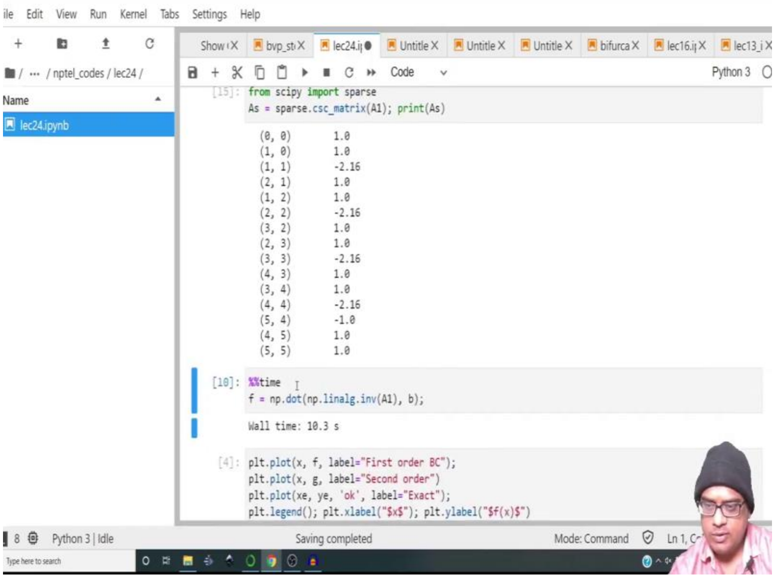
Next topic

scipy.sparse.csc_matrix.__len__

Quick search

So, what it does is instead of storing all the information, it will simply store the data and it will store which row and column it belongs to. So, it helps us introducing the memory overhead. So, I will pass the numpy matrix A 1 to this and let me just assign it to A s, meaning a sparse. Let me then go ahead and print A s. Scipy has no attributes sparse ok.

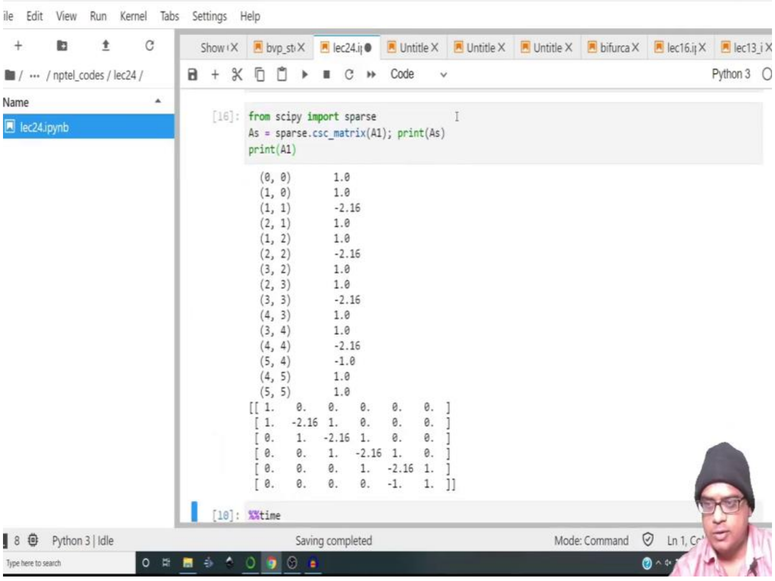
(Refer Slide Time: 05:31)



```
[15]: from scipy import sparse
      As = sparse.csc_matrix(A1); print(As)

        (0, 0)        1.0
        (1, 0)        1.0
        (1, 1)        -2.16
        (2, 1)        1.0
        (1, 2)        1.0
        (2, 2)        -2.16
        (3, 2)        1.0
        (2, 3)        1.0
        (3, 3)        -2.16
        (4, 3)        1.0
        (3, 4)        1.0
        (4, 4)        -2.16
        (5, 4)        -1.0
        (4, 5)        1.0
        (5, 5)        1.0

[10]: %%time
      f = np.dot(np.linalg.inv(A1), b);

      Wall time: 10.3 s

 [4]: plt.plot(x, f, label="First order BC");
      plt.plot(x, g, label="Second order")
      plt.plot(xe, ye, 'ok', label="Exact");
      plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")
```

If I think need to from scipy import sparse yeah ok. So, this is how the matrix is stored.

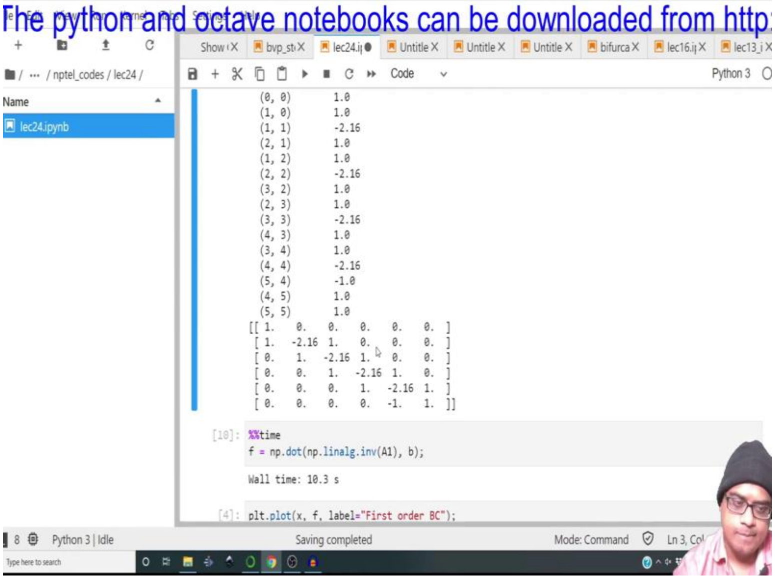
(Refer Slide Time: 05:49)



```
[16]: from scipy import sparse
      As = sparse.csc_matrix(A1); print(As)
      print(A1)

  (0, 0)	1.0
  (1, 0)	1.0
  (1, 1)	-2.16
  (2, 1)	1.0
  (1, 2)	1.0
  (2, 2)	-2.16
  (3, 2)	1.0
  (2, 3)	1.0
  (3, 3)	-2.16
  (4, 3)	1.0
  (3, 4)	1.0
  (4, 4)	-2.16
  (5, 4)	-1.0
  (4, 5)	1.0
  (5, 5)	1.0
[[ 1.    0.    0.    0.    0.    0.  ]
 [ 1.   -2.16  1.    0.    0.    0.  ]
 [ 0.    1.   -2.16  1.    0.    0.  ]
 [ 0.    0.    1.   -2.16  1.    0.  ]
 [ 0.    0.    0.    1.   -2.16  1.  ]
 [ 0.    0.    0.    0.   -1.    1.  ]]

[10]: %%time
```

And just for your reference, I will also print out what A 1 was.

(Refer Slide Time: 05:53)

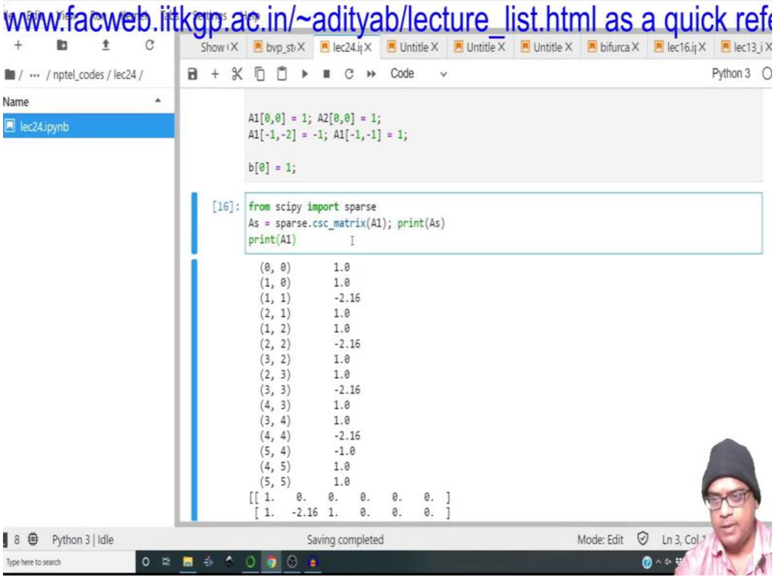


The python and octave notebooks can be downloaded from http:

```
  (0, 0)	1.0
  (1, 0)	1.0
  (1, 1)	-2.16
  (2, 1)	1.0
  (1, 2)	1.0
  (2, 2)	-2.16
  (3, 2)	1.0
  (2, 3)	1.0
  (3, 3)	-2.16
  (4, 3)	1.0
  (3, 4)	1.0
  (4, 4)	-2.16
  (5, 4)	-1.0
  (4, 5)	1.0
  (5, 5)	1.0
[[ 1.    0.    0.    0.    0.    0.  ]
 [ 1.   -2.16  1.    0.    0.    0.  ]
 [ 0.    1.   -2.16  1.    0.    0.  ]
 [ 0.    0.    1.   -2.16  1.    0.  ]
 [ 0.    0.    0.    1.   -2.16  1.  ]
 [ 0.    0.    0.    0.   -1.    1.  ]]

[10]: %%time
      f = np.dot(np.linalg.inv(A1), b);

      Wall time: 10.3 s

[4]:  plt.plot(x, f, label="First order BC");
```

So, look 0, 0 is 1; 0, 0 is 1; 1, 0 is 1; 1, 0 is 1; 1, 1 is this. So, there you go. It is a very efficient way to representing the matrix, and other elements are 0.

(Refer Slide Time: 06:12)



```
A1[0,0] = 1; A2[0,0] = 1;
A1[-1,-2] = -1; A1[-1,-1] = 1;

b[0] = 1;
```

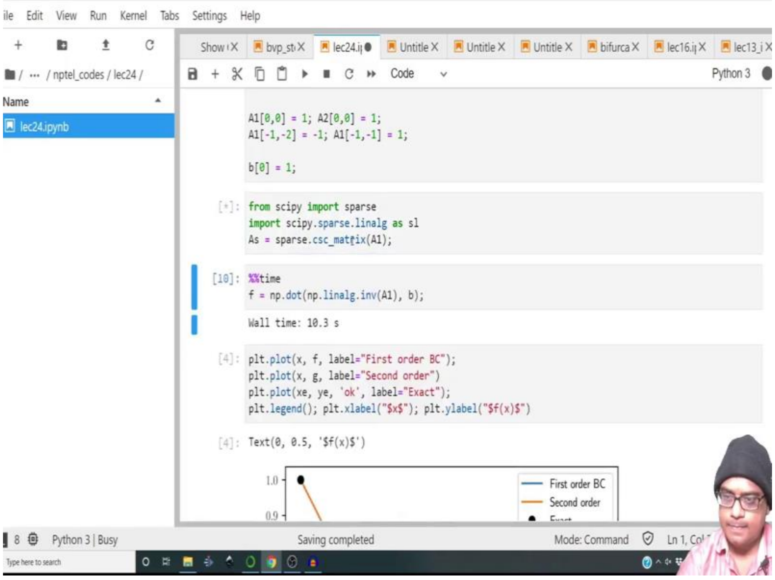
```
[16]: from scipy import sparse
      As = sparse.csc_matrix(A1); print(As)
      print(A1)
```

```
  (0, 0)	1.0
  (1, 0)	1.0
  (1, 1)	-2.16
  (2, 1)	1.0
  (1, 2)	1.0
  (2, 2)	-2.16
  (3, 2)	1.0
  (2, 3)	1.0
  (3, 3)	-2.16
  (4, 3)	1.0
  (3, 4)	1.0
  (4, 4)	-2.16
  (5, 4)	-1.0
  (4, 5)	1.0
  (5, 5)	1.0
[[ 1.    0.    0.    0.    0.    0.  ]
 [ 1.   -2.16  1.    0.    0.    0.  ]
```

And the point is once the matrix becomes larger, there are the number of zeros are going to grow by N^2, by an order of N^2 because most of the things lie on the matrix and lie on the diagonal. And the diagonal can be interpreted as being a line in an area right.

So, as the size increases, the number of zeros increases quadratically rather than linearly. So, why do we bother with converting to this csc format? So, once we have converted this, we can use these sparse matrix algorithms to find the inverse more efficiently.

(Refer Slide Time: 06:50)



```
A1[0,0] = 1; A2[0,0] = 1;
A1[-1,-2] = -1; A1[-1,-1] = 1;

b[0] = 1;
```

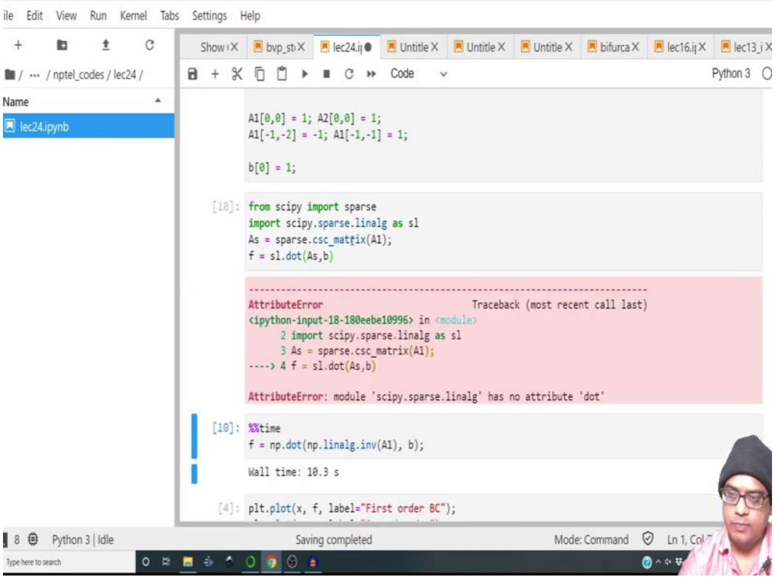
```
[*]: from scipy import sparse
     import scipy.sparse.linalg as sl
     As = sparse.csc_matrix(A1);
```

```
[10]: %%time
      f = np.dot(np.linalg.inv(A1), b);

      Wall time: 10.3 s
```

```
[4]: plt.plot(x, f, label="First order BC");
     plt.plot(x, g, label="Second order")
     plt.plot(xe, ye, 'ok', label="Exact");
     plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")

[4]: Text(0, 0.5, '$f(x)$')
```

So, let me import it alright. So, we have imported this sparse.linalg module and now, we are going to take a dot product. So, A s is already of csc format.

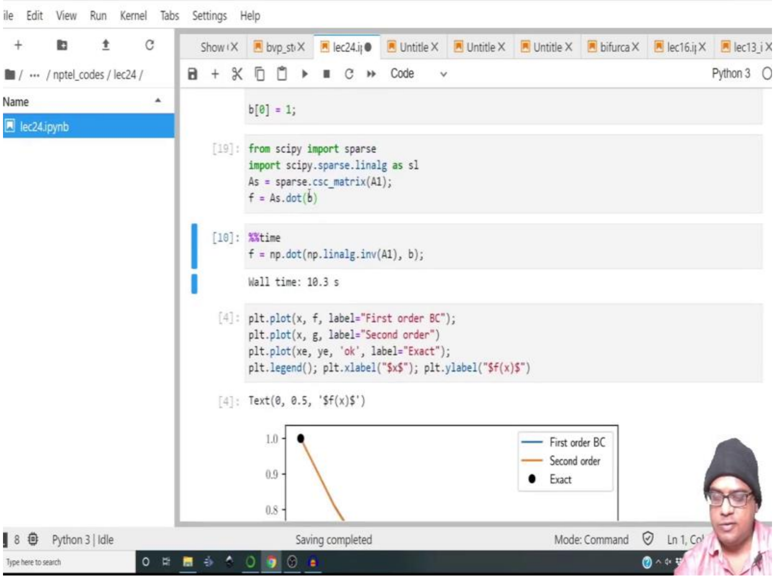
(Refer Slide Time: 07:34)



```
A1[0,0] = 1; A2[0,0] = 1;
A1[-1,-2] = -1; A1[-1,-1] = 1;

b[0] = 1;

[18]: from scipy import sparse
      import scipy.sparse.linalg as sl
      As = sparse.csc_matrix(A1);
      f = sl.dot(As,b)

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-18-180eebe10996> in <module>
      2 import scipy.sparse.linalg as sl
      3 As = sparse.csc_matrix(A1);
----> 4 f = sl.dot(As,b)

AttributeError: module 'scipy.sparse.linalg' has no attribute 'dot'

[10]: %%time
      f = np.dot(np.linalg.inv(A1), b);

      Wall time: 10.3 s

[4]:  plt.plot(x, f, label="First order BC");
```

So, now we can write f = s l.dot A s , b sorry.

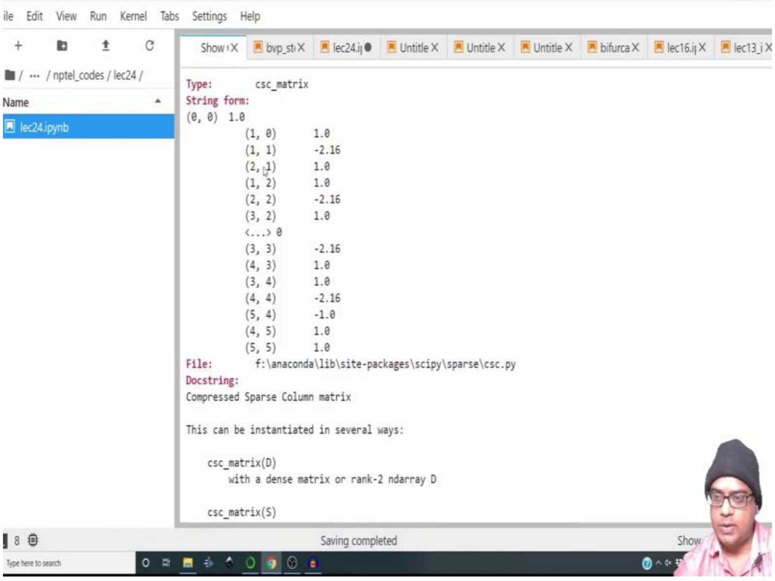
(Refer Slide Time: 07:49)



```
b[0] = 1;

[19]: from scipy import sparse
      import scipy.sparse.linalg as sl
      As = sparse.csc_matrix(A1);
      f = As.dot(b)

[10]: %%time
      f = np.dot(np.linalg.inv(A1), b);

      Wall time: 10.3 s

[4]:  plt.plot(x, f, label="First order BC");
      plt.plot(x, g, label="Second order")
      plt.plot(xe, ye, 'ok', label="Exact");
      plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")

[4]:  Text(0, 0.5, '$f(x)$')
```

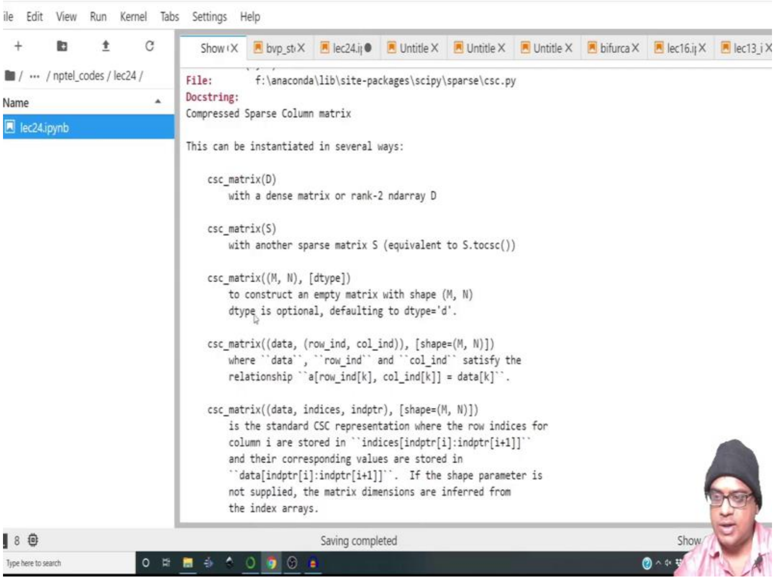
So, it should be A s .dot b. So, that is the way of achieving this dot product ok.

(Refer Slide Time: 07:59)



```
Type:        csc_matrix
String form:
(0, 0)  1.0
          (1, 0)      1.0
          (1, 1)      -2.16
          (2, 1)      1.0
          (1, 2)      1.0
          (2, 2)      -2.16
          (3, 2)      1.0
          <...> 0
          (3, 3)      -2.16
          (4, 3)      1.0
          (3, 4)      1.0
          (4, 4)      -2.16
          (5, 4)      -1.0
          (4, 5)      1.0
          (5, 5)      1.0
File:        f:\anaconda\lib\site-packages\scipy\sparse\csc.py
Docstring:
Compressed Sparse Column matrix

This can be instantiated in several ways:

    csc_matrix(D)
        with a dense matrix or rank-2 ndarray D

    csc_matrix(S)
```

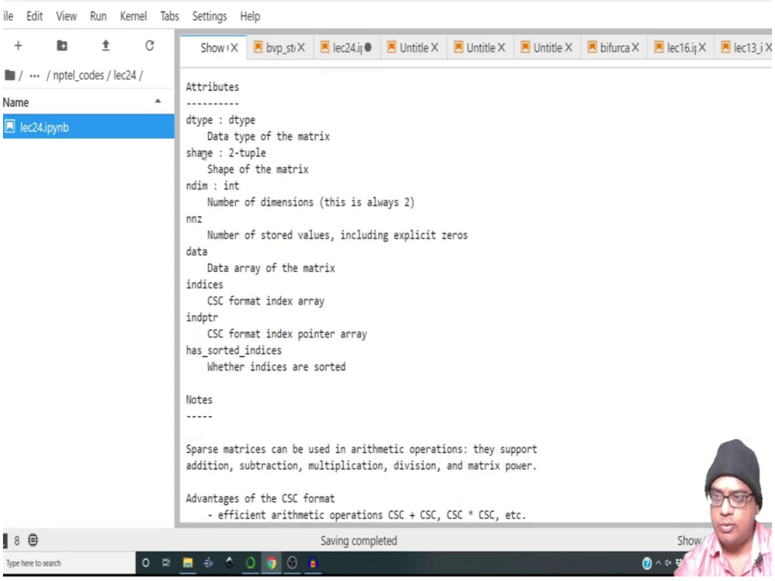
(Refer Slide Time: 08:00)



```
File:        f:\anaconda\lib\site-packages\scipy\sparse\csc.py
Docstring:
Compressed Sparse Column matrix

This can be instantiated in several ways:

    csc_matrix(D)
        with a dense matrix or rank-2 ndarray D

    csc_matrix(S)
        with another sparse matrix S (equivalent to S.tocsc())

    csc_matrix((M, N), [dtype])
        to construct an empty matrix with shape (M, N)
        dtype is optional, defaulting to dtype='d'.

    csc_matrix((data, (row_ind, col_ind)), [shape=(M, N)])
        where ``data``, ``row_ind`` and ``col_ind`` satisfy the
        relationship ``a[row_ind[k], col_ind[k]] = data[k]``.

    csc_matrix((data, indices, indptr), [shape=(M, N)])
        is the standard CSC representation where the row indices for
        column i are stored in ``indices[indptr[i]:indptr[i+1]]``
        and their corresponding values are stored in
        ``data[indptr[i]:indptr[i+1]]``.  If the shape parameter is
        not supplied, the matrix dimensions are inferred from
        the index arrays.
```

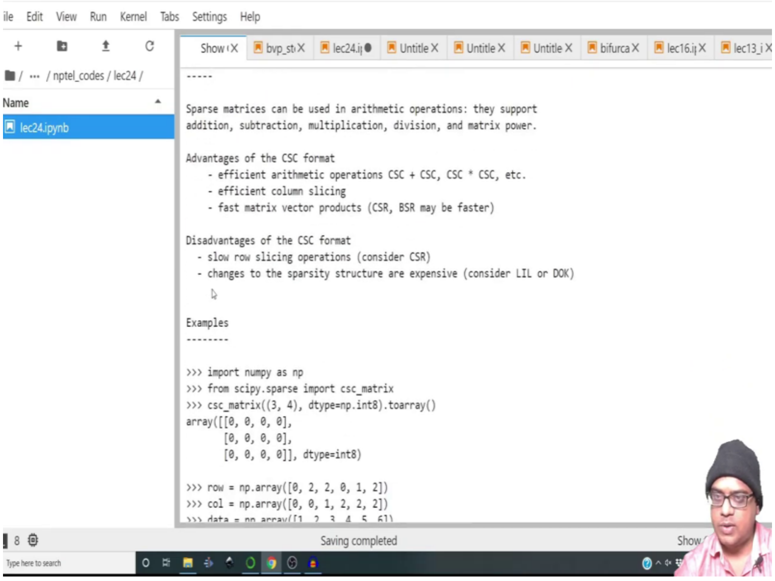
So, A s has an attribute, it is a csc matrix and it has certain, not attributes.

(Refer Slide Time: 08:02)



```
Attributes
----------
dtype : dtype
    Data type of the matrix
shape : 2-tuple
    Shape of the matrix
ndim : int
    Number of dimensions (this is always 2)
nnz
    Number of stored values, including explicit zeros
data
    Data array of the matrix
indices
    CSC format index array
indptr
    CSC format index pointer array
has_sorted_indices
    Whether indices are sorted

Notes
-----

Sparse matrices can be used in arithmetic operations: they support
addition, subtraction, multiplication, division, and matrix power.

Advantages of the CSC format
    - efficient arithmetic operations CSC + CSC, CSC * CSC, etc.
```

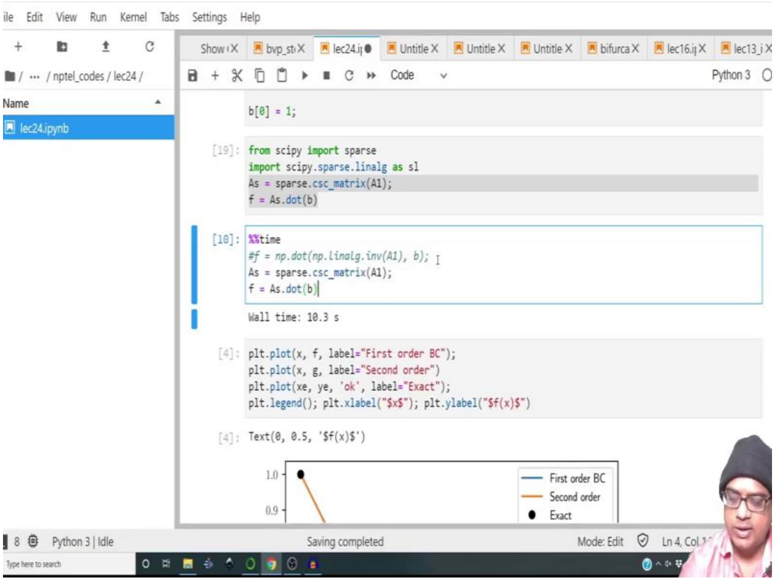
(Refer Slide Time: 08:06)



```
-----

Sparse matrices can be used in arithmetic operations: they support
addition, subtraction, multiplication, division, and matrix power.

Advantages of the CSC format
    - efficient arithmetic operations CSC + CSC, CSC * CSC, etc.
    - efficient column slicing
    - fast matrix vector products (CSR, BSR may be faster)

Disadvantages of the CSC format
  - slow row slicing operations (consider CSR)
  - changes to the sparsity structure are expensive (consider LIL or DOK)


Examples
--------

>>> import numpy as np
>>> from scipy.sparse import csc_matrix
>>> csc_matrix((3, 4), dtype=np.int8).toarray()
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]], dtype=int8)

>>> row = np.array([0, 2, 2, 0, 1, 2])
>>> col = np.array([0, 0, 1, 2, 2, 2])
>>> data = np.array([1, 2, 3, 4, 5, 6])
```

But some additional functions associated with it which include the dotting with a vector ok. It is not mentioned in this contextual help. But you can take a dot products with something like this. So, A s .dot will give you the required dot product and so, what we will do is we will take this bit of code.

(Refer Slide Time: 08:36)



```
b[0] = 1;
```

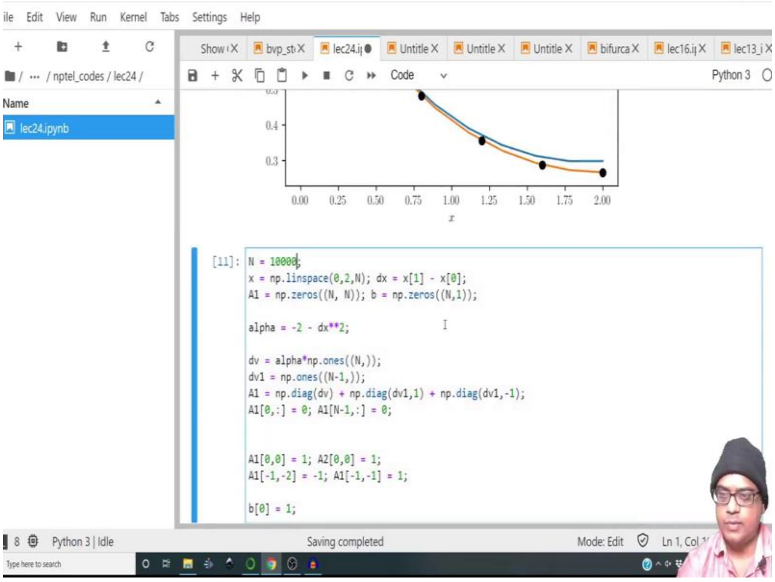
```
[19]: from scipy import sparse
      import scipy.sparse.linalg as sl
      As = sparse.csc_matrix(A1);
      f = As.dot(b)
```

```
[10]: %%time
      #f = np.dot(np.linalg.inv(A1), b);
      As = sparse.csc_matrix(A1);
      f = As.dot(b)

      Wall time: 10.3 s
```

```
[4]:  plt.plot(x, f, label="First order BC");
      plt.plot(x, g, label="Second order")
      plt.plot(xe, ye, 'ok', label="Exact");
      plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")

[4]:  Text(0, 0.5, '$f(x)$')
```

And let me copy this. Let me comment out this old program, let me paste this and let me see how much time it takes.

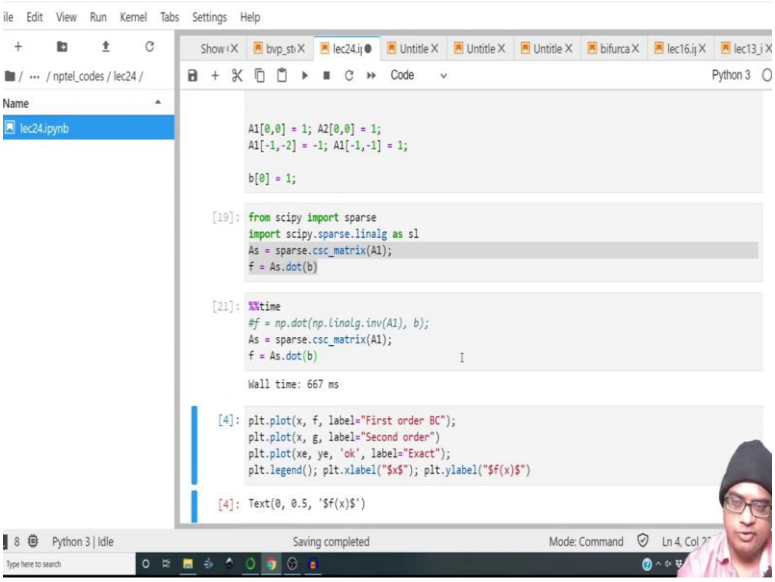
(Refer Slide Time: 08:44)



```
[11]: N = 10000;
      x = np.linspace(0,2,N); dx = x[1] - x[0];
      A1 = np.zeros((N, N)); b = np.zeros((N,1));

      alpha = -2 - dx**2;

      dv = alpha*np.ones((N,));
      dv1 = np.ones((N-1,));
      A1 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
      A1[0,:] = 0; A1[N-1,:] = 0;


      A1[0,0] = 1; A2[0,0] = 1;
      A1[-1,-2] = -1; A1[-1,-1] = 1;

      b[0] = 1;
```

So, 10000 was the number that we had used. So, let me run this.

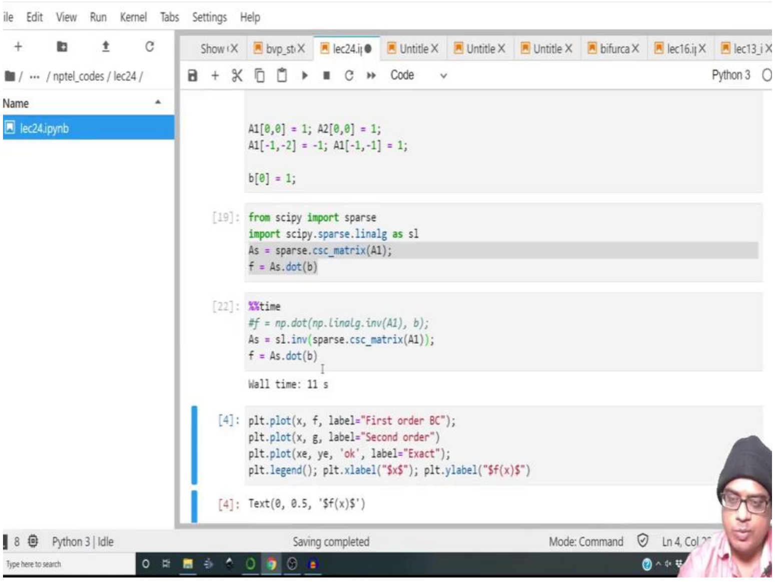
(Refer Slide Time: 08:52)


ile Edit View Run Kernel Tabs Settings Help
/ ··· / nptel_codes / lec24 /
Name
lec24.ipynb
Show (bvp_st lec24.i Untitle Untitle Untitle bifurca lec16.i lec13_i
Code
Python 3
8 Python 3 | Idle
Saving completed
Mode: Command
Ln 4, Col 2
Type here to search

```
A1[0,0] = 1; A2[0,0] = 1;
A1[-1,-2] = -1; A1[-1,-1] = 1;

b[0] = 1;

[19]: from scipy import sparse
      import scipy.sparse.linalg as sl
      As = sparse.csc_matrix(A1);
      f = As.dot(b)

[21]: %%time
      #f = np.dot(np.linalg.inv(A1), b);
      As = sparse.csc_matrix(A1);
      f = As.dot(b)

      Wall time: 667 ms

[4]: plt.plot(x, f, label="First order BC");
     plt.plot(x, g, label="Second order")
     plt.plot(xe, ye, 'ok', label="Exact");
     plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")

[4]: Text(0, 0.5, '$f(x)$')
```

So, for 10000, it took 10 seconds; so let me see how much time, it takes say 667 milliseconds. So, obviously, the sparse solver runs much faster than the numpy inversion alright. So, we forgot to take the inverse ok. So, probably that is why it runs so much faster. I forgot to completely take the inverse.

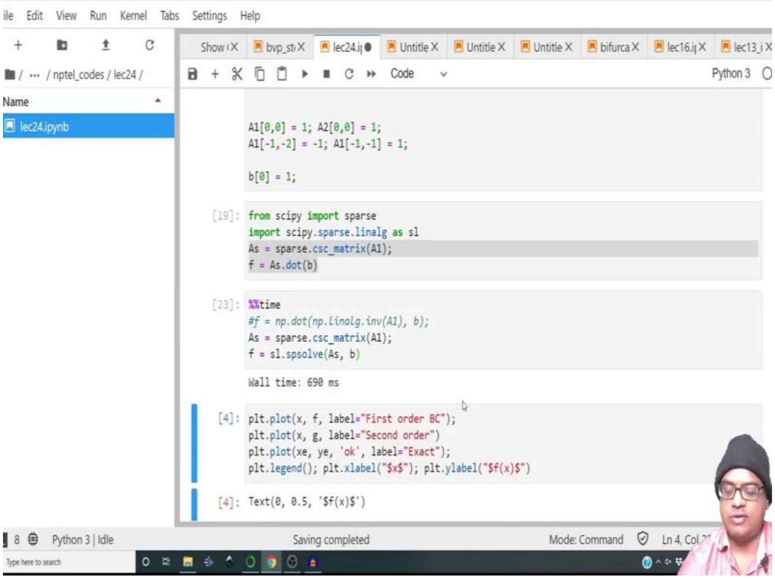
(Refer Slide Time: 09:17)


ile Edit View Run Kernel Tabs Settings Help
/ ··· / nptel_codes / lec24 /
Name
lec24.ipynb
Show (bvp_st lec24.i Untitle Untitle Untitle bifurca lec16.i lec13_i
Code
Python 3
8 Python 3 | Idle
Saving completed
Mode: Command
Ln 4, Col 2
Type here to search

```
A1[0,0] = 1; A2[0,0] = 1;
A1[-1,-2] = -1; A1[-1,-1] = 1;

b[0] = 1;

[19]: from scipy import sparse
      import scipy.sparse.linalg as sl
      As = sparse.csc_matrix(A1);
      f = As.dot(b)

[22]: %%time
      #f = np.dot(np.linalg.inv(A1), b);
      As = sl.inv(sparse.csc_matrix(A1));
      f = As.dot(b)

      Wall time: 11 s

[4]: plt.plot(x, f, label="First order BC");
     plt.plot(x, g, label="Second order")
     plt.plot(xe, ye, 'ok', label="Exact");
     plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")

[4]: Text(0, 0.5, '$f(x)$')
```

So, s l dot inv is the way to do the inverse ok. So, let me run this now. So, the (Refer Time: 09:27) was 10 seconds for numpy. Let us see how much time it takes for scipy inverse. Maybe it is not, it took 11 seconds. So, it took much more time. So, it took the same amount of time and it is not clear as to why this has happened this way; but it should have happened faster, maybe for much more larger matrices. But something which will run faster will be the solver. So, let me get rid of this.

(Refer Slide Time: 10:07)



```
A1[0,0] = 1; A2[0,0] = 1;
A1[-1,-2] = -1; A1[-1,-1] = 1;

b[0] = 1;
```

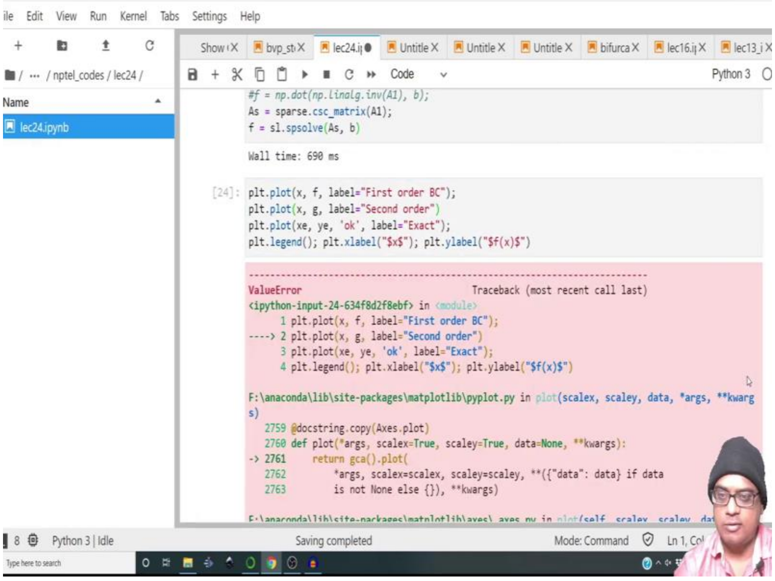
```
[19]: from scipy import sparse
      import scipy.sparse.linalg as sl
      As = sparse.csc_matrix(A1);
      f = As.dot(b)
```

```
[23]: %%time
      #f = np.dot(np.linalg.inv(A1), b);
      As = sparse.csc_matrix(A1);
      f = sl.spsolve(As, b)

      Wall time: 690 ms
```

```
[4]: plt.plot(x, f, label="First order BC");
     plt.plot(x, g, label="Second order")
     plt.plot(xe, ye, 'ok', label="Exact");
     plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")

[4]: Text(0, 0.5, '$f(x)$')
```

In fact, let me make it sparse. We need it to be sparse and so, there is a solver called as s1.sp solve. So, sp solve stands for sparse solve, it lies inside the sparse dot linear algebra sub module and it takes inputs as the sparse matrix in csc format and the matrix b. So, let me run this. This should run much faster ok.

(Refer Slide Time: 10:46)



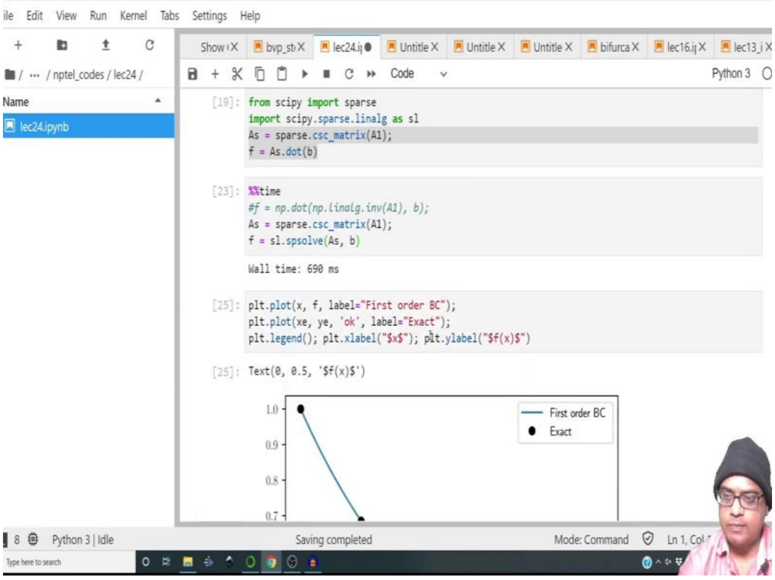
```
      #f = np.dot(np.linalg.inv(A1), b);
      As = sparse.csc_matrix(A1);
      f = sl.spsolve(As, b)

      Wall time: 690 ms
```

```
[24]: plt.plot(x, f, label="First order BC");
      plt.plot(x, g, label="Second order")
      plt.plot(xe, ye, 'ok', label="Exact");
      plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-24-634f8d2f8ebf> in <module>
      1 plt.plot(x, f, label="First order BC");
----> 2 plt.plot(x, g, label="Second order")
      3 plt.plot(xe, ye, 'ok', label="Exact");
      4 plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")

F:\anaconda\lib\site-packages\matplotlib\pyplot.py in plot(scalex, scaley, data, *args, **kwarg
s)
   2759 @docstring.copy(Axes.plot)
   2760 def plot(*args, scalex=True, scaley=True, data=None, **kwargs):
-> 2761     return gca().plot(
   2762         *args, scalex=scalex, scaley=scaley, **({"data": data} if data
   2763         is not None else {}), **kwargs)

F:\anaconda\lib\site-packages\matplotlib\axes\_axes.py in plot(self, scalex, scaley, dat
```

So, that took 690 milliseconds. So, yeah, let us plot it and see whether it is fine or not. There is a bunch of errors and what is error? x and y.

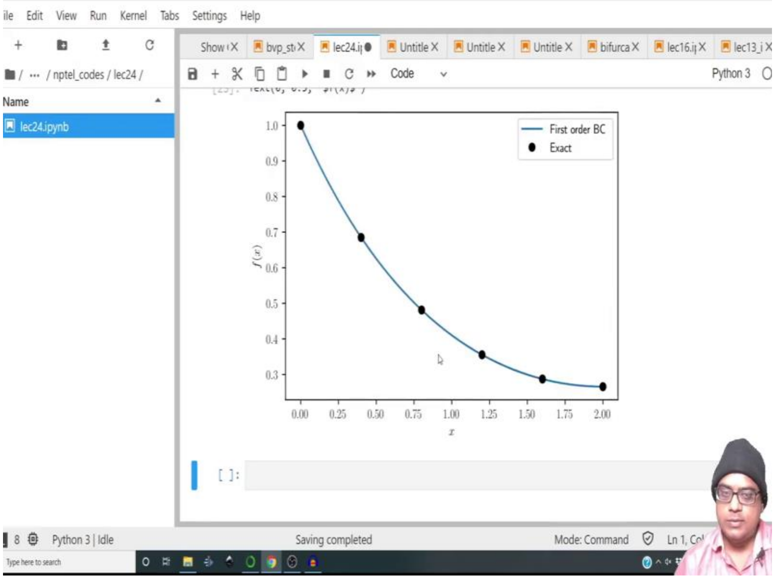
(Refer Slide Time: 11:01)



```
[19]: from scipy import sparse
      import scipy.sparse.linalg as sl
      As = sparse.csc_matrix(A1);
      f = As.dot(b)

[23]: %%time
      #f = np.dot(np.linalg.inv(A1), b);
      As = sparse.csc_matrix(A1);
      f = sl.spsolve(As, b)

      Wall time: 690 ms

[25]: plt.plot(x, f, label="First order BC");
      plt.plot(xe, ye, 'ok', label="Exact");
      plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")

[25]: Text(0, 0.5, '$f(x)$')
```

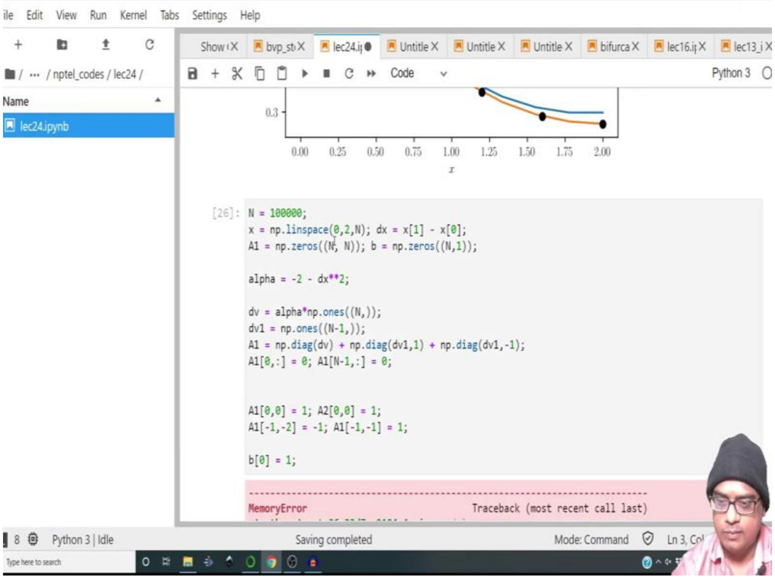
So, let me get rid of g, we do not need g for now. Yeah, ok.

(Refer Slide Time: 11:05)



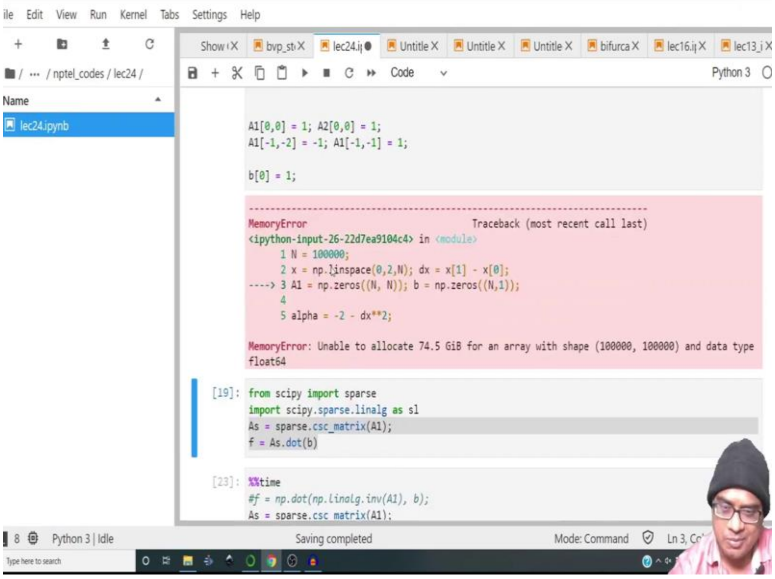
So, with 10000 points everything matches quite well ok. Let me reduce the number of points.

(Refer Slide Time: 11:14)



```
[26]: N = 100000;
      x = np.linspace(0,2,N); dx = x[1] - x[0];
      A1 = np.zeros((N, N)); b = np.zeros((N,1));

      alpha = -2 - dx**2;

      dv = alpha*np.ones((N,));
      dv1 = np.ones((N-1,));
      A1 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
      A1[0,:] = 0; A1[N-1,:] = 0;


      A1[0,0] = 1; A2[0,0] = 1;
      A1[-1,-2] = -1; A1[-1,-1] = 1;

      b[0] = 1;

      ---------------------------------------------------------------------------
      MemoryError                               Traceback (most recent call last)
```

(Refer Slide Time: 11:17)

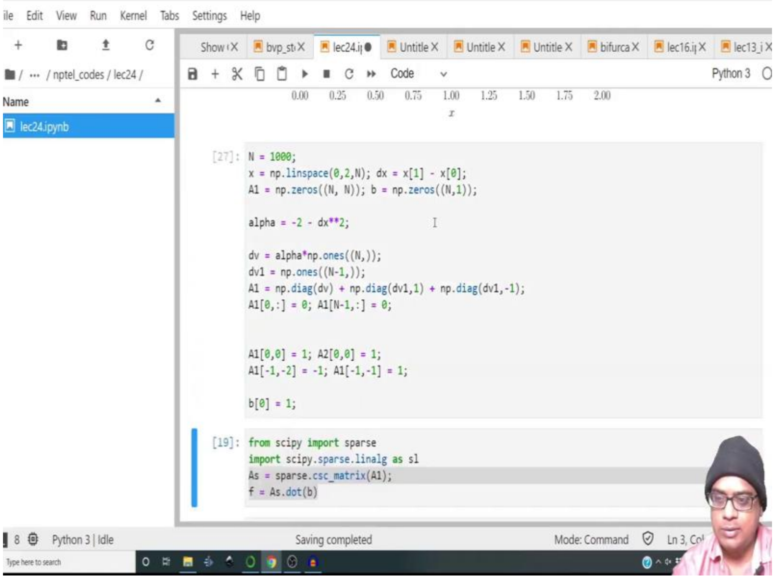


```
      A1[0,0] = 1; A2[0,0] = 1;
      A1[-1,-2] = -1; A1[-1,-1] = 1;

      b[0] = 1;

      ---------------------------------------------------------------------------
      MemoryError                               Traceback (most recent call last)
      <ipython-input-26-22d7ea9104c4> in <module>
            1 N = 100000;
            2 x = np.linspace(0,2,N); dx = x[1] - x[0];
      ----> 3 A1 = np.zeros((N, N)); b = np.zeros((N,1));
            4
            5 alpha = -2 - dx**2;

      MemoryError: Unable to allocate 74.5 GiB for an array with shape (100000, 100000) and data type
      float64

[19]: from scipy import sparse
      import scipy.sparse.linalg as sl
      As = sparse.csc_matrix(A1);
      f = As.dot(b)

[23]: %%time
      #f = np.dot(np.linalg.inv(A1), b);
      As = sparse.csc_matrix(A1);
```

In fact, let me increase it by a million and this is memory error ok; unable to allocate 74.5 GiB. The series might be 32 gigabytes. So, this is where Sc outperforms, you do not really need to worry. But if you define A to be originally a sparse matrix, you can avoid this error. So, the fact is I am creating a numpy array out of this and that it its initializing a whole bunch of zeros, where you do not really need to initialize zeros.

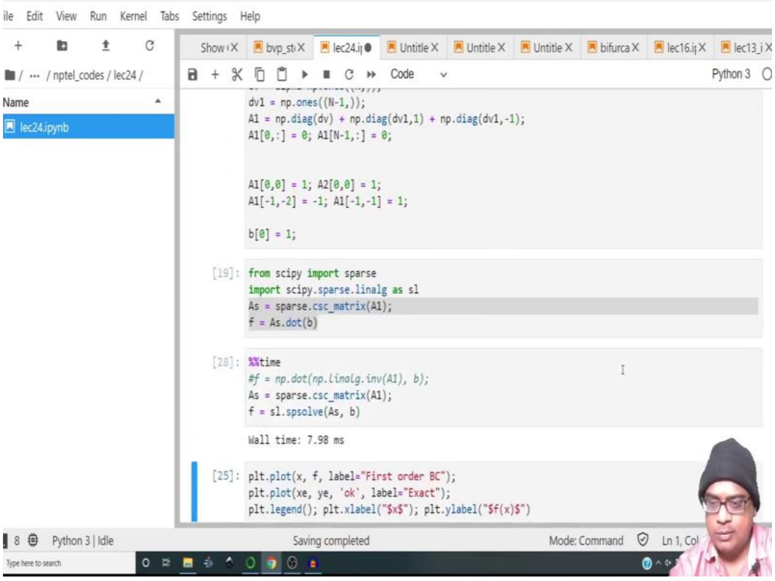
And once we do things in PETSc, it will be clear how you can avoid this model. So, PETSc is much more efficient in this regard. I am sure you could substitute or rather declare the A 1 matrix to be a bunch of zeros rather than declaring them as a bunch of zeros, you could simply define it as a sparse matrix, where everything else is 0 by default.

(Refer Slide Time: 12:15)



```
[27]: N = 1000;
      x = np.linspace(0,2,N); dx = x[1] - x[0];
      A1 = np.zeros((N, N)); b = np.zeros((N,1));

      alpha = -2 - dx**2;

      dv = alpha*np.ones((N,));
      dv1 = np.ones((N-1,));
      A1 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
      A1[0,:] = 0; A1[N-1,:] = 0;


      A1[0,0] = 1; A2[0,0] = 1;
      A1[-1,-2] = -1; A1[-1,-1] = 1;

      b[0] = 1;

[19]: from scipy import sparse
      import scipy.sparse.linalg as sl
      As = sparse.csc_matrix(A1);
      f = As.dot(b)
```

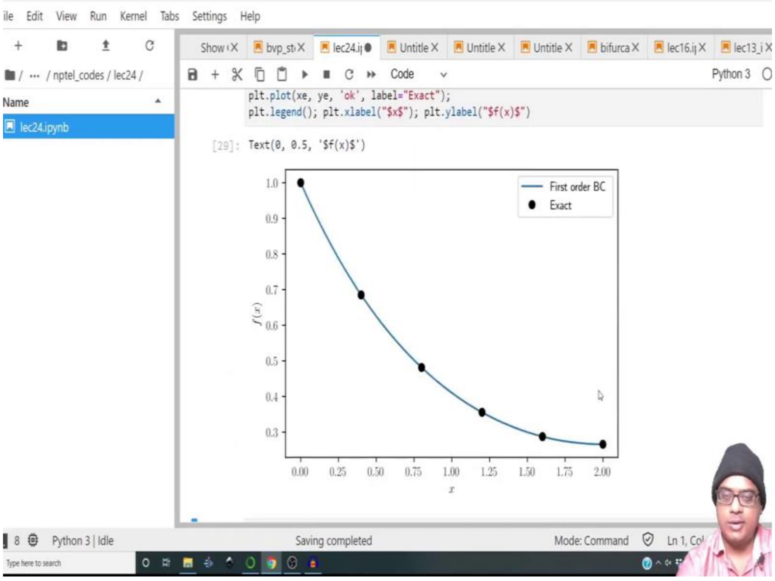
(Refer Slide Time: 12:19)



ile Edit View Run Kernel Tabs Settings Help
/ ··· / nptel_codes / lec24 /
Name
lec24.ipynb
Show ‹ bvp_st lec24.i Untitle Untitle Untitle bifurca lec16.i lec13_i
Code
Python 3
8 Python 3 | Idle
Saving completed
Mode: Command
Ln 1, Col
Type here to search

```
      dv1 = np.ones((N-1,));
      A1 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
      A1[0,:] = 0; A1[N-1,:] = 0;


      A1[0,0] = 1; A2[0,0] = 1;
      A1[-1,-2] = -1; A1[-1,-1] = 1;

      b[0] = 1;

[19]: from scipy import sparse
      import scipy.sparse.linalg as sl
      As = sparse.csc_matrix(A1);
      f = As.dot(b)

[28]: %%time
      #f = np.dot(np.linalg.inv(A1), b);
      As = sparse.csc_matrix(A1);
      f = sl.spsolve(As, b)

      Wall time: 7.98 ms

[25]: plt.plot(x, f, label="First order BC");
      plt.plot(xe, ye, 'ok', label="Exact");
      plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")
```

(Refer Slide Time: 12:23)

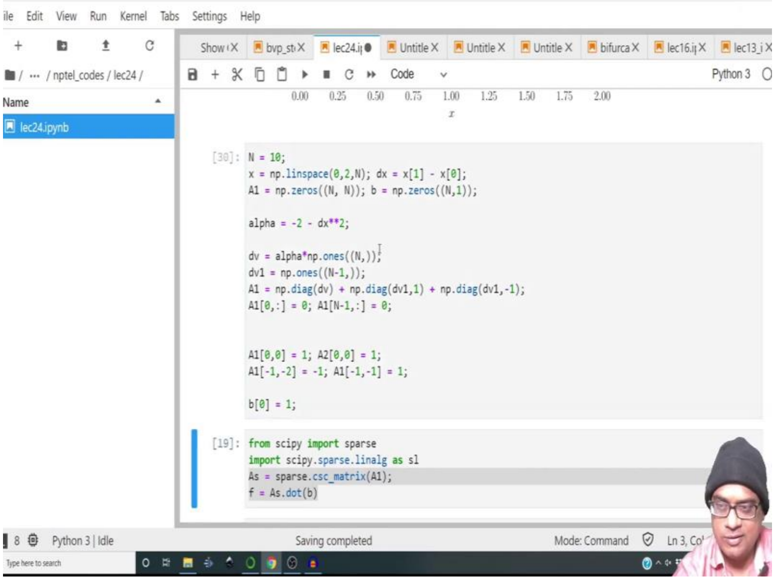


```
plt.plot(xe, ye, 'ok', label="Exact");
plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")
```

```
[29]: Text(0, 0.5, '$f(x)$')
```

So, let me dial it down 1000, yeah almost 8 milliseconds to plot it, great.

(Refer Slide Time: 12:32)

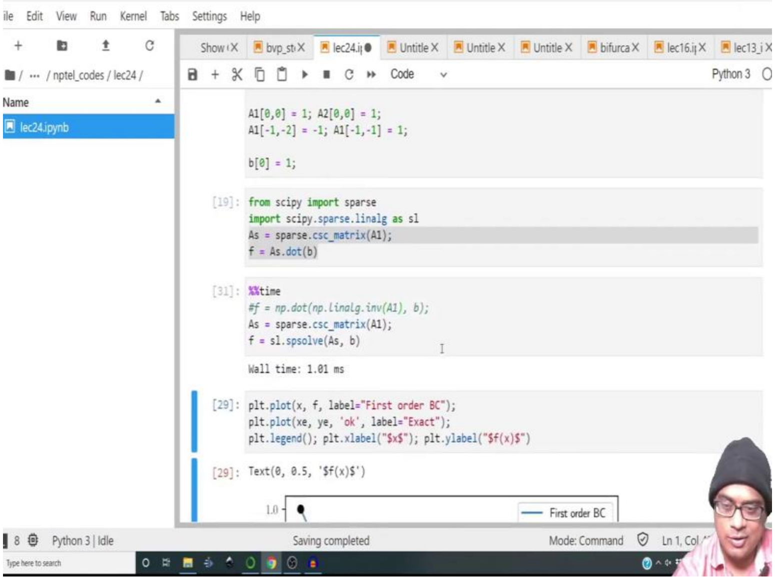


```
[30]: N = 10;
      x = np.linspace(0,2,N); dx = x[1] - x[0];
      A1 = np.zeros((N, N)); b = np.zeros((N,1));

      alpha = -2 - dx**2;

      dv = alpha*np.ones((N,));
      dv1 = np.ones((N-1,));
      A1 = np.diag(dv) + np.diag(dv1,1) + np.diag(dv1,-1);
      A1[0,:] = 0; A1[N-1,:] = 0;


      A1[0,0] = 1; A2[0,0] = 1;
      A1[-1,-2] = -1; A1[-1,-1] = 1;

      b[0] = 1;
```

```
[19]: from scipy import sparse
      import scipy.sparse.linalg as sl
      As = sparse.csc_matrix(A1);
      f = As.dot(b)
```

So, let me make it 10, just to show that it is giving that old error.

(Refer Slide Time: 12:36)



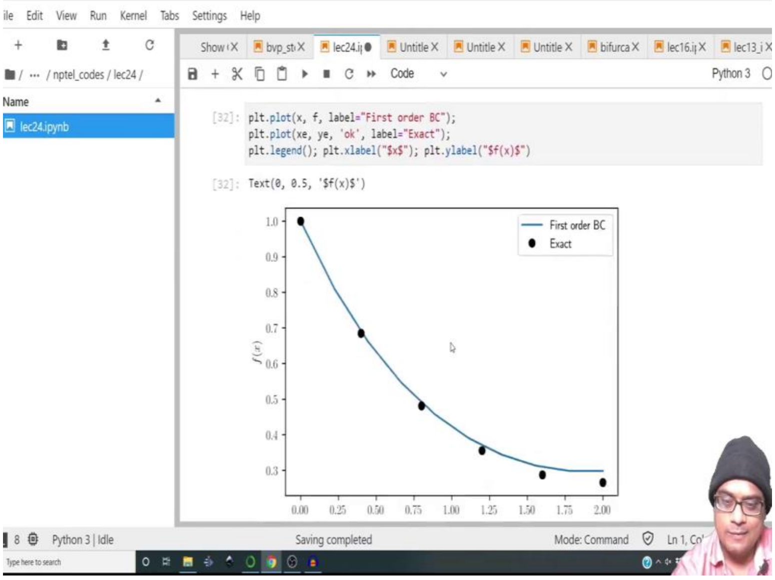
```
A1[0,0] = 1; A2[0,0] = 1;
A1[-1,-2] = -1; A1[-1,-1] = 1;

b[0] = 1;
```

```
[19]: from scipy import sparse
      import scipy.sparse.linalg as sl
      As = sparse.csc_matrix(A1);
      f = As.dot(b)
```

```
[31]: %%time
      #f = np.dot(np.linalg.inv(A1), b);
      As = sparse.csc_matrix(A1);
      f = sl.spsolve(As, b)

      Wall time: 1.01 ms
```

```
[29]: plt.plot(x, f, label="First order BC");
      plt.plot(xe, ye, 'ok', label="Exact");
      plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")

[29]: Text(0, 0.5, '$f(x)$')
```

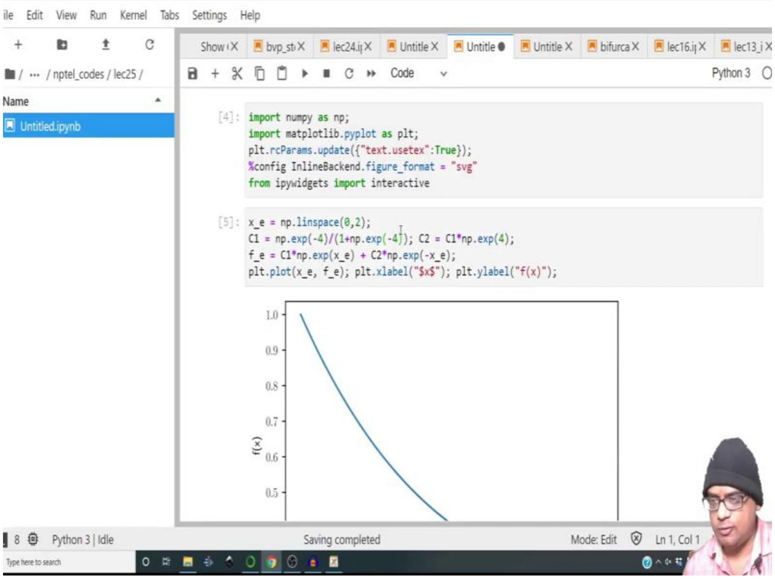
(Refer Slide Time: 12:37)



```
[32]: plt.plot(x, f, label="First order BC");
      plt.plot(xe, ye, 'ok', label="Exact");
      plt.legend(); plt.xlabel("$x$"); plt.ylabel("$f(x)$")

[32]: Text(0, 0.5, '$f(x)$')
```

That is that you imagine because of the first order nature of the boundary condition. So, yeah, I mean with this in mind, we go to the next part of this lecture which is to use the inbuilt functions of scipy to solve boundary value problems.

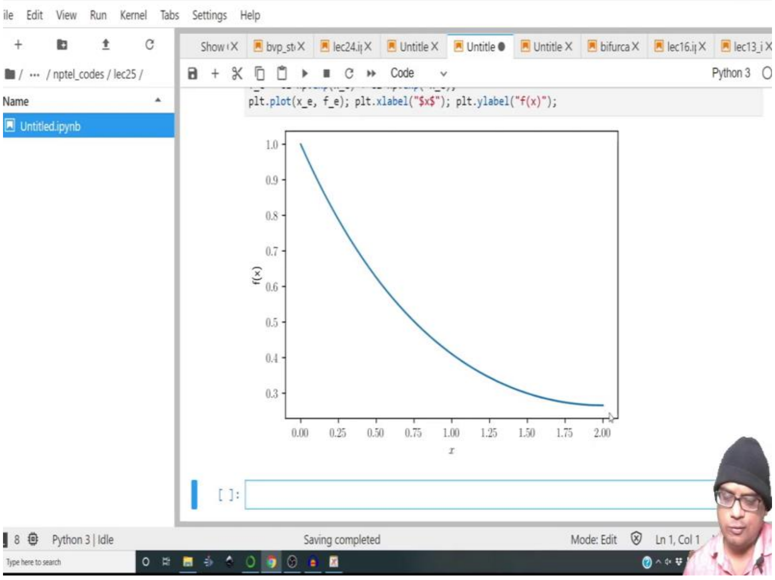
(Refer Slide Time: 12:59)



```
[4]: import numpy as np;
     import matplotlib.pyplot as plt;
     plt.rcParams.update({"text.usetex":True});
     %config InlineBackend.figure_format = "svg"
     from ipywidgets import interactive

[5]: x_e = np.linspace(0,2);
     C1 = np.exp(-4)/(1+np.exp(-4)); C2 = C1*np.exp(4);
     f_e = C1*np.exp(x_e) + C2*np.exp(-x_e);
     plt.plot(x_e, f_e); plt.xlabel("$x$"); plt.ylabel("f(x)");
```

So, let us continue our journey. So, again, I have created a new file. I have imported the usual things and I have created the analytical solution of the problem that we had considered in the previous lecture.

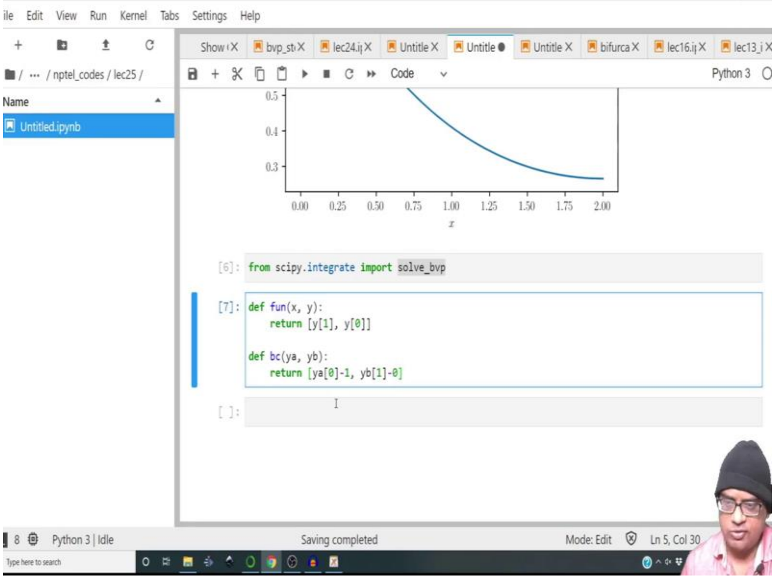
(Refer Slide Time: 13:15)



```
plt.plot(x_e, f_e); plt.xlabel("$x$"); plt.ylabel("f(x)");

[ ]:
```

And the solution looks something like this. At x = 0, it is 1; at x = 2, the slope is = z. So, we made a nice finite difference program and we were able to you solve it. And once we have done that, we were able to show how we can implement the second order solution with the cost node. So, now we are going to use the inbuilt function of scipy to solve the ODE.

(Refer Slide Time: 13:47)



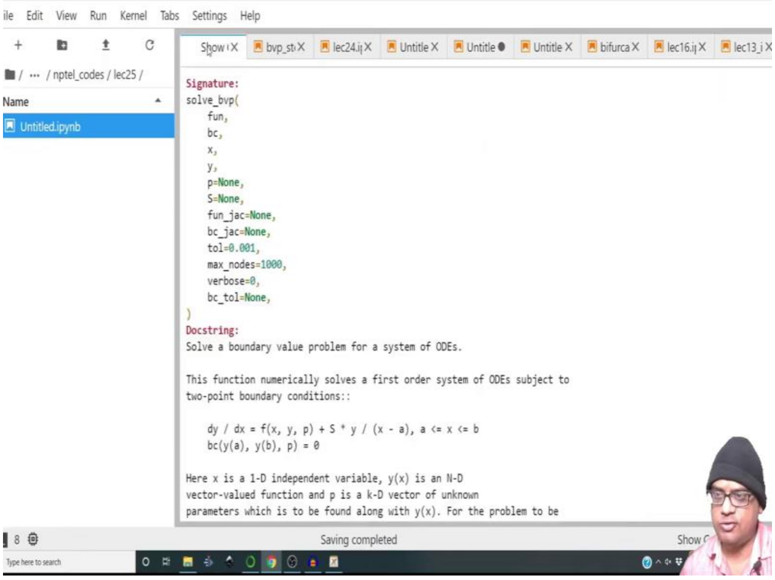
```
[6]: from scipy.integrate import solve_bvp
```

```
[7]: def fun(x, y):
         return [y[1], y[0]]

     def bc(ya, yb):
         return [ya[0]-1, yb[1]-0]
```

```
[ ]:
```

So, first things first let us import another from scipy. integrate, let us import solve bvp. So, like solve ivp, there is another function called a solve bvp. It stands for solve boundary value problem alright. So, let me import this. So, what does this function contains?

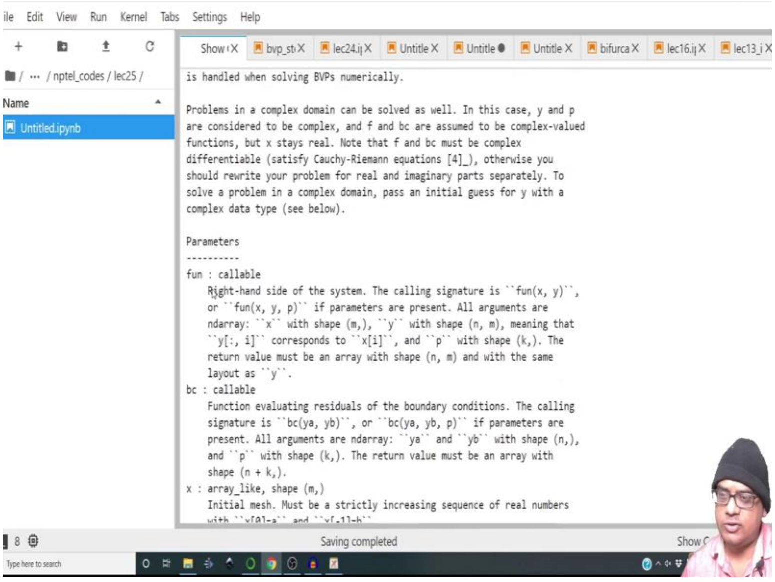
(Refer Slide Time: 14:16)



```
Signature:
solve_bvp(
    fun,
    bc,
    x,
    y,
    p=None,
    S=None,
    fun_jac=None,
    bc_jac=None,
    tol=0.001,
    max_nodes=1000,
    verbose=0,
    bc_tol=None,
)
Docstring:
Solve a boundary value problem for a system of ODEs.

This function numerically solves a first order system of ODEs subject to
two-point boundary conditions::

    dy / dx = f(x, y, p) + S * y / (x - a), a <= x <= b
    bc(y(a), y(b), p) = 0

Here x is a 1-D independent variable, y(x) is an N-D
vector-valued function and p is a k-D vector of unknown
parameters which is to be found along with y(x). For the problem to be
```

So, let us have a look at the contextual help. So, it takes as an input the function the boundary condition the x array, it is the domain; the y which is the solution array, p stands for parameters, function Jacobian, boundary condition Jacobian, tolerance, max nodes, verbose blah blah blah.

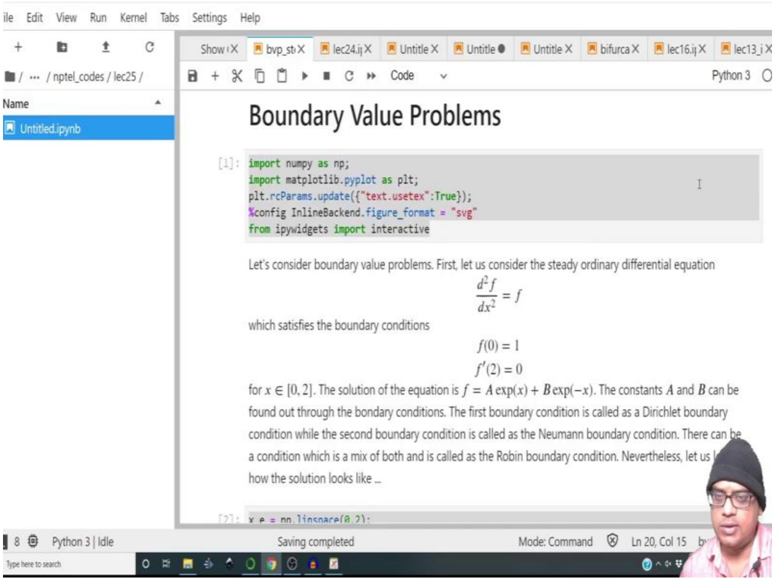
(Refer Slide Time: 14:33)



```
is handled when solving BVPs numerically.

Problems in a complex domain can be solved as well. In this case, y and p
are considered to be complex, and f and bc are assumed to be complex-valued
functions, but x stays real. Note that f and bc must be complex
differentiable (satisfy Cauchy-Riemann equations [4]_), otherwise you
should rewrite your problem for real and imaginary parts separately. To
solve a problem in a complex domain, pass an initial guess for y with a
complex data type (see below).

Parameters
----------
fun : callable
    Right-hand side of the system. The calling signature is ``fun(x, y)``,
    or ``fun(x, y, p)`` if parameters are present. All arguments are
    ndarray: ``x`` with shape (m,), ``y`` with shape (n, m), meaning that
    ``y[:, i]`` corresponds to ``x[i]``, and ``p`` with shape (k,). The
    return value must be an array with shape (n, m) and with the same
    layout as ``y``.
bc : callable
    Function evaluating residuals of the boundary conditions. The calling
    signature is ``bc(ya, yb)``, or ``bc(ya, yb, p)`` if parameters are
    present. All arguments are ndarray: ``ya`` and ``yb`` with shape (n,),
    and ``p`` with shape (k,). The return value must be an array with
    shape (n + k,).
x : array_like, shape (m,)
    Initial mesh. Must be a strictly increasing sequence of real numbers
    with ``x[0]=a`` and ``x[-1]=b``.
```

So, function is callable ok. So, let us create the function. So, def function and it will take inputs as x and y and it will return something. So, let us go to the previous program right.

(Refer Slide Time: 15:00)



Boundary Value Problems

```
[1]: import numpy as np;
     import matplotlib.pyplot as plt;
     plt.rcParams.update({"text.usetex":True});
     %config InlineBackend.figure_format = "svg"
     from ipywidgets import interactive
```

Let's consider boundary value problems. First, let us consider the steady ordinary differential equation

$$\frac{d^2 f}{dx^2} = f$$

which satisfies the boundary conditions

$$f(0) = 1$$

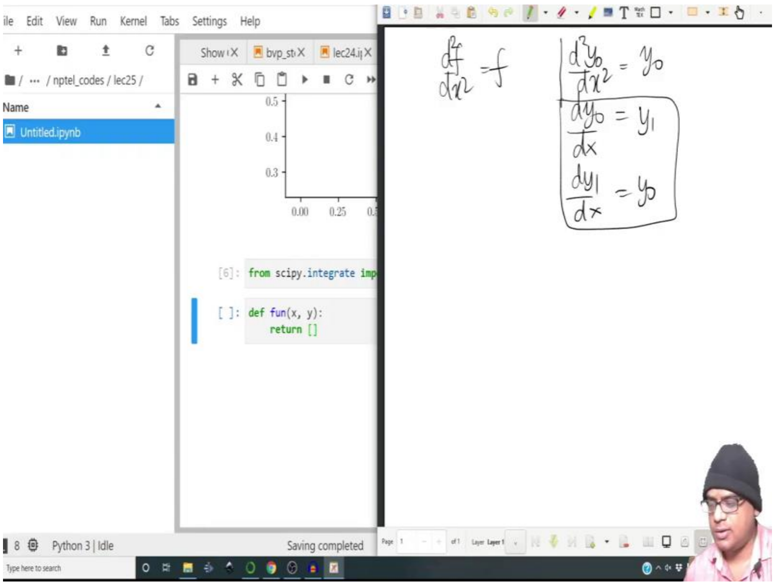
$$f'(2) = 0$$

for $x \in [0, 2]$. The solution of the equation is $f = A\exp(x) + B\exp(-x)$. The constants A and B can be found out through the bondary conditions. The first boundary condition is called as a Dirichlet boundary condition while the second boundary condition is called as the Neumann boundary condition. There can be a condition which is a mix of both and is called as the Robin boundary condition. Nevertheless, let us how the solution looks like ...

```
[2]: x_e = np.linspace(0,2);
```

In fact, let me show you what the equation was. It was $\frac{d^2 f}{dx^2} = f$.

(Refer Slide Time: 15:14)



```
[6]: from scipy.integrate imp
```

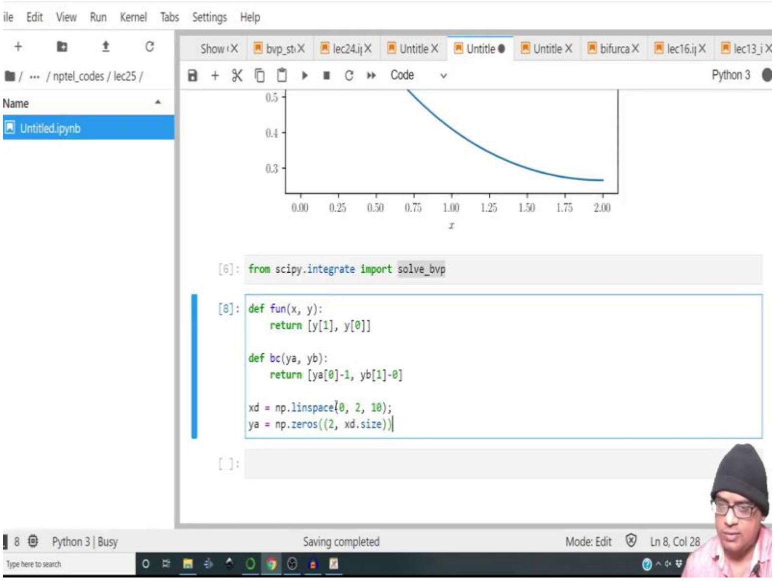
```
[ ]: def fun(x, y):
         return []
```

So, we have to first split the problem into two into a series of first order equations. So, let me instead of f, let me call it $\frac{d^2 y_0}{dx^2} = y_0$. Let $\frac{dy_0}{dx} = y_1$. This implies $\frac{dy_1}{dx} = y_0$ and this is a very simple splitting of the equation. This particular equation into these two particular equations alright.

So, we have to return what y_1 and y_0 fine. So, I will go over here, I will return y_1 and I will return over here y_0 alright. Now, I have to define what? I have to define the boundary condition. So, define the boundary condition. So, here the boundary conditions are implemented as y $_a$ and y $_b$ and a, b are identified as the sort of end points of the domain.

So, I will the inputs to this will be y $_a$, y $_b$ and I will return the following. So, the boundary condition was y(0) = 2, y' or rather y is 0 was = 1 and y' (2) was = 0. So, y $_a$ is = 1 essentially and y_b' is = 0. So, the way to implement this is you have to give the residue. So, I must give over here ya[0]-1. So, 0 stands for the Dirichlet and here, it will be yb[1]-0 which stands for the Neumann condition alright.

(Refer Slide Time: 17:16)



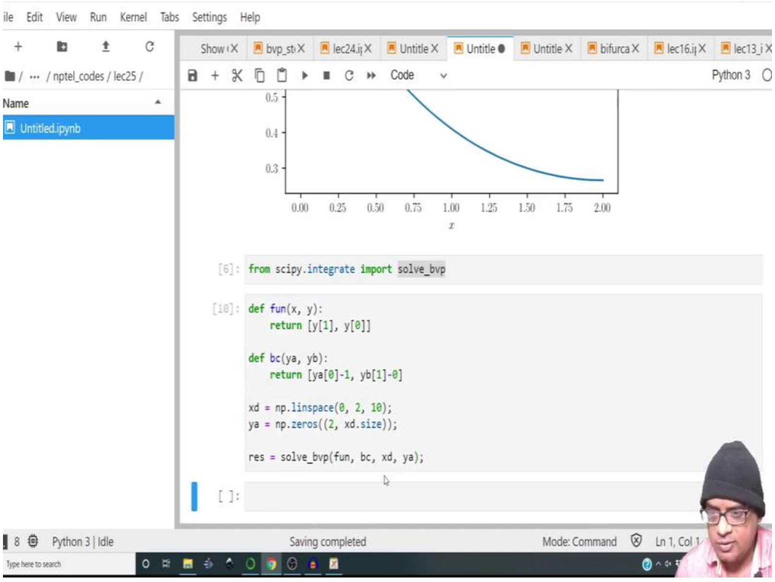
```
[6]: from scipy.integrate import solve_bvp
```

```
[8]: def fun(x, y):
         return [y[1], y[0]]

     def bc(ya, yb):
         return [ya[0]-1, yb[1]-0]

     xd = np.linspace(0, 2, 10);
     ya = np.zeros((2, xd.size))
```

```
[ ]:
```

So, after this, we must create the domain. So, xd = np.linspace(0, 2, 10) andya = np.zeros((2, xd.size)). So, it will be simply xd.size and this also works, fine.

(Refer Slide Time: 17:55)



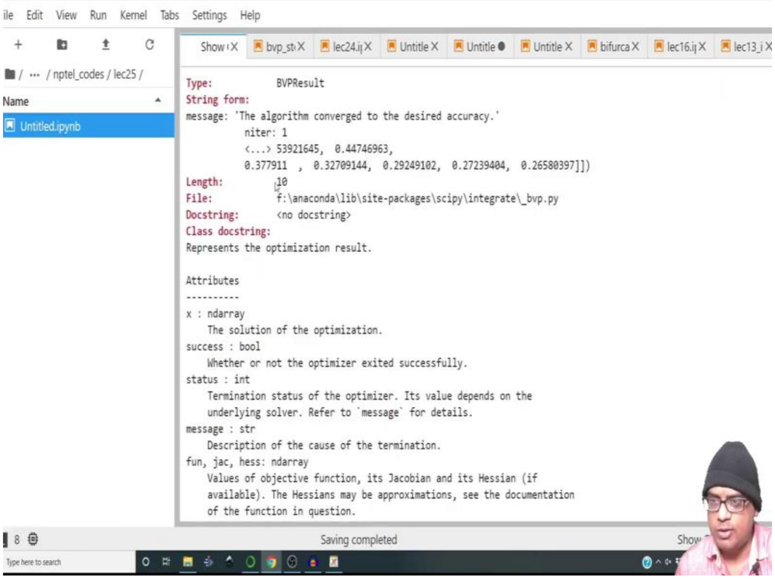
```
[6]: from scipy.integrate import solve_bvp
```

```
[10]: def fun(x, y):
          return [y[1], y[0]]

      def bc(ya, yb):
          return [ya[0]-1, yb[1]-0]

      xd = np.linspace(0, 2, 10);
      ya = np.zeros((2, xd.size));

      res = solve_bvp(fun, bc, xd, ya);
```

```
[ ]:
```

Now, what we will do is we will call the function. So, residue of this function, let me call it res solve_bvp. We will pass the function, we will pass the. So, what did we have to pass? Let me function boundary condition x, y; function boundary condition x, y. So, yeah that acts as the initial guess that is pretty much it. So, let me run this to see if there is an error. There is no error.

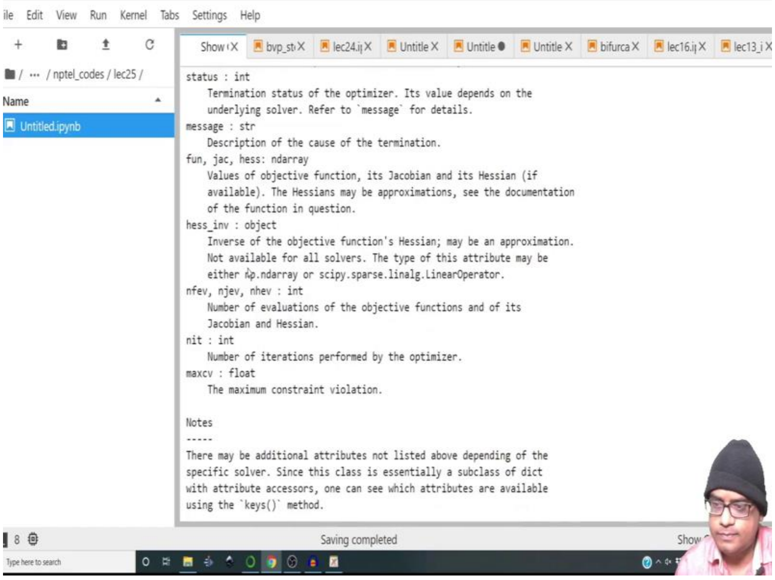
(Refer Slide Time: 18:35)



```
Type:           BVPResult
String form:
message: 'The algorithm converged to the desired accuracy.'
           niter: 1
           <...> 53921645,  0.44746963,
           0.377911  ,  0.32709144,  0.29249102,  0.27239404,  0.26580397]])
Length:         10
File:           f:\anaconda\lib\site-packages\scipy\integrate\_bvp.py
Docstring:      <no docstring>
Class docstring:
Represents the optimization result.

Attributes
----------
x : ndarray
    The solution of the optimization.
success : bool
    Whether or not the optimizer exited successfully.
status : int
    Termination status of the optimizer. Its value depends on the
    underlying solver. Refer to `message` for details.
message : str
    Description of the cause of the termination.
fun, jac, hess: ndarray
    Values of objective function, its Jacobian and its Hessian (if
    available). The Hessians may be approximations, see the documentation
    of the function in question.
```

So, now what does res contain? Let us see if we can have a contextual help on it.

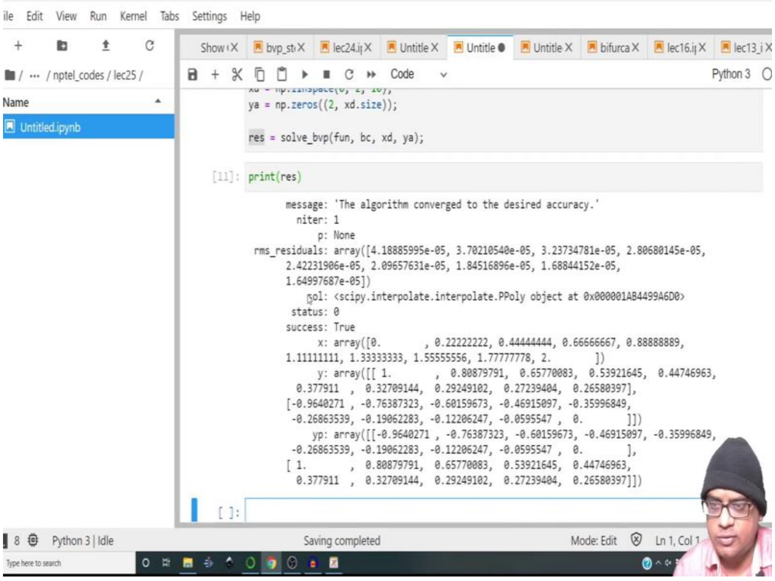
(Refer Slide Time: 18:37)



```
status : int
    Termination status of the optimizer. Its value depends on the
    underlying solver. Refer to `message` for details.
message : str
    Description of the cause of the termination.
fun, jac, hess: ndarray
    Values of objective function, its Jacobian and its Hessian (if
    available). The Hessians may be approximations, see the documentation
    of the function in question.
hess_inv : object
    Inverse of the objective function's Hessian; may be an approximation.
    Not available for all solvers. The type of this attribute may be
    either np.ndarray or scipy.sparse.linalg.LinearOperator.
nfev, njev, nhev : int
    Number of evaluations of the objective functions and of its
    Jacobian and Hessian.
nit : int
    Number of iterations performed by the optimizer.
maxcv : float
    The maximum constraint violation.

Notes
-----
There may be additional attributes not listed above depending of the
specific solver. Since this class is essentially a subclass of dict
with attribute accessors, one can see which attributes are available
using the `keys()` method.
```

So, x contains all these things, essentially it treats it as an optimization problem, which is fine ok.

(Refer Slide Time: 18:55)



ile Edit View Run Kernel Tabs Settings Help
/ ··· / nptel_codes / lec25 /
Name
Untitled.ipynb
Show (bvp_st lec24.i Untitle Untitle Untitle bifurca lec16.i lec13_i
Code
Python 3
8 Python 3 | Idle
Saving completed
Mode: Edit
Ln 1, Col 1
Type here to search

```
ya = np.zeros((2, xd.size));

res = solve_bvp(fun, bc, xd, ya);
```

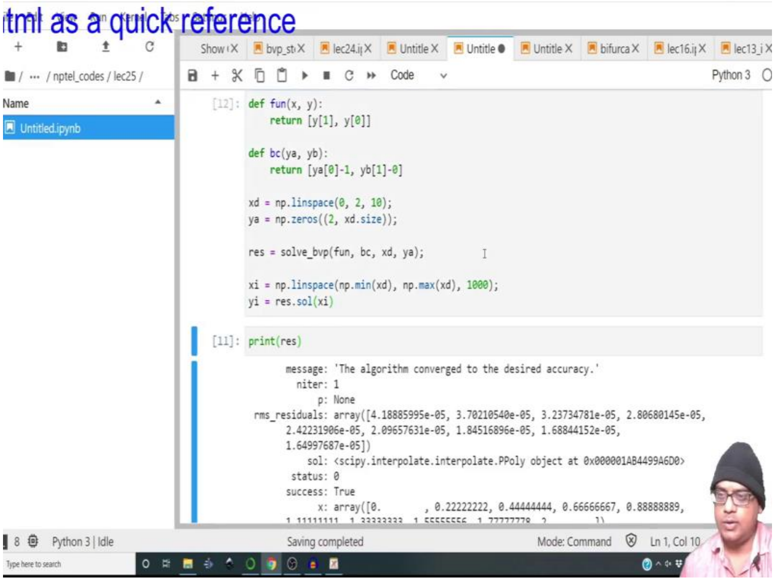
```
[11]: print(res)
```

```
      message: 'The algorithm converged to the desired accuracy.'
        niter: 1
            p: None
 rms_residuals: array([4.18885995e-05, 3.70210540e-05, 3.23734781e-05, 2.80680145e-05,
      2.42231906e-05, 2.09657631e-05, 1.84516896e-05, 1.68844152e-05,
      1.64997687e-05])
          sol: <scipy.interpolate.interpolate.PPoly object at 0x000001AB4499A6D0>
       status: 0
      success: True
            x: array([0.        , 0.22222222, 0.44444444, 0.66666667, 0.88888889,
      1.11111111, 1.33333333, 1.55555556, 1.77777778, 2.        ])
            y: array([[ 1.        ,  0.80879791,  0.65770083,  0.53921645,  0.44746963,
        0.377911  ,  0.32709144,  0.29249102,  0.27239404,  0.26580397],
      [-0.9640271 , -0.76387323, -0.60159673, -0.46915097, -0.35996849,
       -0.26863539, -0.19062283, -0.12206247, -0.0595547 ,  0.        ]])
           yp: array([[-0.9640271 , -0.76387323, -0.60159673, -0.46915097, -0.35996849,
       -0.26863539, -0.19062283, -0.12206247, -0.0595547 ,  0.        ],
      [ 1.        ,  0.80879791,  0.65770083,  0.53921645,  0.44746963,
        0.377911  ,  0.32709144,  0.29249102,  0.27239404,  0.26580397]])
```

```
[ ]:
```

So, let me print res over here. So, what does it contain? Iterations, xp array, yp array. So, yp is a it is a double array yeah. Sol, so we are more interested in sol; sol is the interpolation object. And I think we have used this previously as well whenever you have access to an interpolation object might as well use that interpolation object. So, how do we go about using it?

(Refer Slide Time: 19:24)



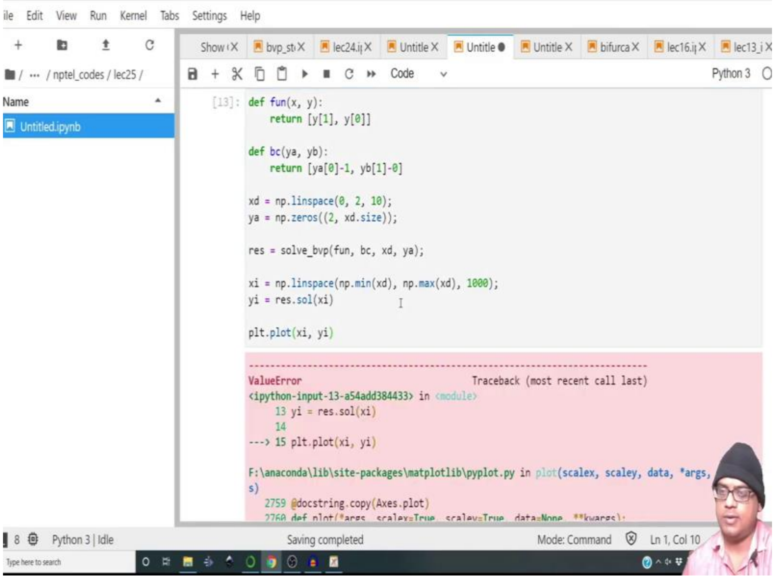
```
[12]: def fun(x, y):
          return [y[1], y[0]]

      def bc(ya, yb):
          return [ya[0]-1, yb[1]-0]

      xd = np.linspace(0, 2, 10);
      ya = np.zeros((2, xd.size));

      res = solve_bvp(fun, bc, xd, ya);

      xi = np.linspace(np.min(xd), np.max(xd), 1000);
      yi = res.sol(xi)
```

```
[11]: print(res)
```

```
      message: 'The algorithm converged to the desired accuracy.'
        niter: 1
            p: None
 rms_residuals: array([4.18885995e-05, 3.70210540e-05, 3.23734781e-05, 2.80680145e-05,
      2.42231906e-05, 2.09657631e-05, 1.84516896e-05, 1.68844152e-05,
      1.64997687e-05])
          sol: <scipy.interpolate.interpolate.PPoly object at 0x000001AB4499A6D0>
       status: 0
      success: True
            x: array([0.        , 0.22222222, 0.44444444, 0.66666667, 0.88888889,
      1.11111111, 1.33333333, 1.55555556, 1.77777778, 2.        ])
```

So, I will create a new grid. So, x interpolate will be np. linspace, this will be min of it will be the minimum of xd to maximum of xd. xd and I will take 1000 points. So, then yi = res.sol(xi)[0], but now that has to be evaluated over xi alright.

(Refer Slide Time: 20:01)

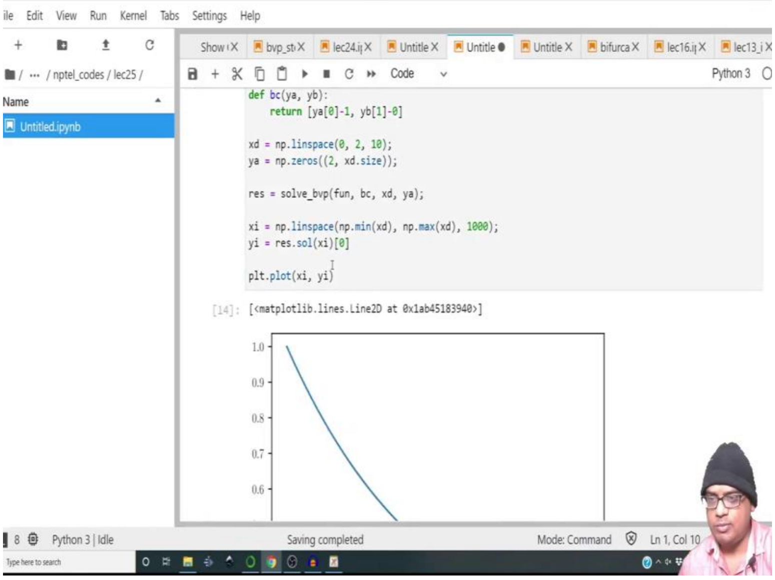


```
[13]: def fun(x, y):
          return [y[1], y[0]]

      def bc(ya, yb):
          return [ya[0]-1, yb[1]-0]

      xd = np.linspace(0, 2, 10);
      ya = np.zeros((2, xd.size));

      res = solve_bvp(fun, bc, xd, ya);

      xi = np.linspace(np.min(xd), np.max(xd), 1000);
      yi = res.sol(xi)

      plt.plot(xi, yi)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-a54add384433> in <module>
     13 yi = res.sol(xi)
     14
---> 15 plt.plot(xi, yi)

F:\anaconda\lib\site-packages\matplotlib\pyplot.py in plot(scalex, scaley, data, *args,
s)
   2759 @docstring.copy(Axes.plot)
   2760 def plot(*args, scalex=True, scaley=True, data=None, **kwargs):
```

So, now, with the help of this, we can simply plot xi , yi. Oops, there is a there is some error ok.

(Refer Slide Time: 20:15)

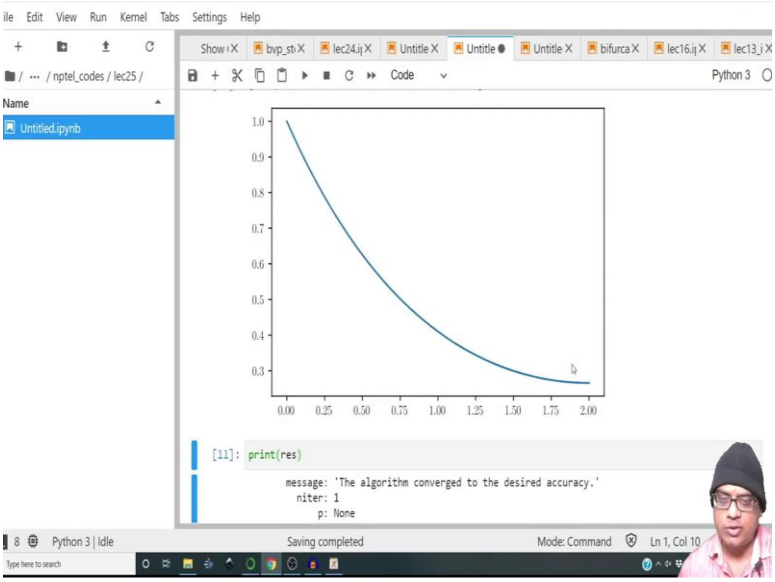


```
def bc(ya, yb):
    return [ya[0]-1, yb[1]-0]

xd = np.linspace(0, 2, 10);
ya = np.zeros((2, xd.size));

res = solve_bvp(fun, bc, xd, ya);

xi = np.linspace(np.min(xd), np.max(xd), 1000);
yi = res.sol(xi)[0]

plt.plot(xi, yi)
```

```
[14]: [<matplotlib.lines.Line2D at 0x1ab45183940>]
```

We just need the residue of the first function that is we need the solution for y_0. Because remember, y_0 was the original function and we made an auxiliary function y_1 to split this into a series of first order equation. So, we are more interested in y_1.

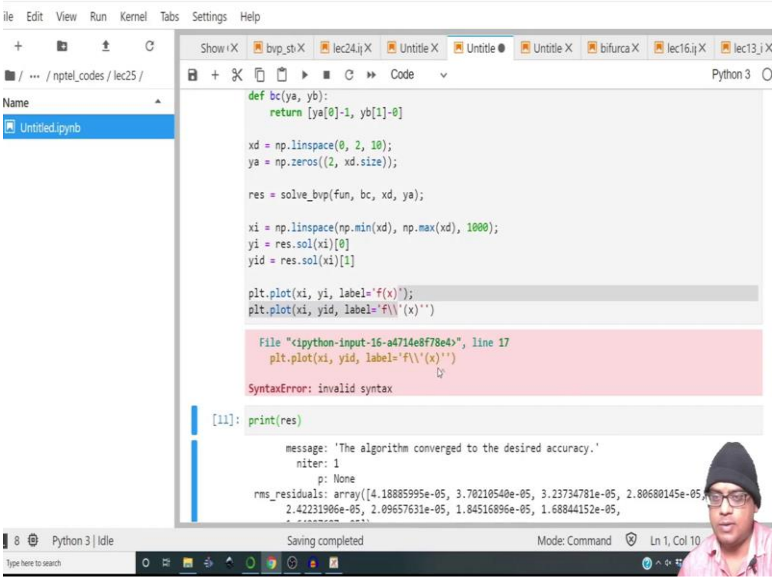
(Refer Slide Time: 20:38)



```
[11]: print(res)

       message: 'The algorithm converged to the desired accuracy.'
         niter: 1
             p: None
```

So, let me plot this boom, there you have it this is the solution that we have been looking for.

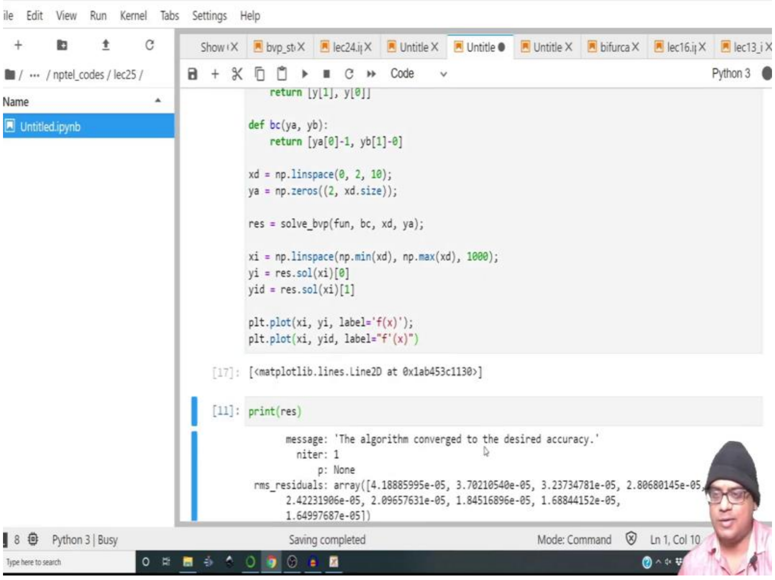
(Refer Slide Time: 20:45)


ile Edit View Run Kernel Tabs Settings Help
Show ‹X bvp_st‹X lec24.i X Untitle X Untitle ● Untitle X bifurca X lec16.i X lec13_i X
/ ··· / nptel_codes / lec25 /
Name
Untitled.ipynb
Code
Python 3
8 Python 3 | Idle
Saving completed
Mode: Command
Ln 1, Col 10
Type here to search

```
def bc(ya, yb):
    return [ya[0]-1, yb[1]-0]

xd = np.linspace(0, 2, 10);
ya = np.zeros((2, xd.size));

res = solve_bvp(fun, bc, xd, ya);

xi = np.linspace(np.min(xd), np.max(xd), 1000);
yi = res.sol(xi)[0]
yid = res.sol(xi)[1]

plt.plot(xi, yi, label='f(x)');
plt.plot(xi, yid, label='f\\'(x)'')

  File "<ipython-input-16-a4714e8f78e4>", line 17
    plt.plot(xi, yid, label='f\\'(x)'')

SyntaxError: invalid syntax

[11]: print(res)

       message: 'The algorithm converged to the desired accuracy.'
         niter: 1
             p: None
  rms_residuals: array([4.18885995e-05, 3.70210540e-05, 3.23734781e-05, 2.80680145e-05,
       2.42231906e-05, 2.09657631e-05, 1.84516896e-05, 1.68844152e-05,
```

In fact, because I have yi, I can also get the yi derivative which will be yid = res.sol(xi)[1] and I can go ahead and plot the derivative as well. Yeah, it is an issue with the little string.

(Refer Slide Time: 21:23)



```
    return [y[1], y[0]]

def bc(ya, yb):
    return [ya[0]-1, yb[1]-0]

xd = np.linspace(0, 2, 10);
ya = np.zeros((2, xd.size));

res = solve_bvp(fun, bc, xd, ya);

xi = np.linspace(np.min(xd), np.max(xd), 1000);
yi = res.sol(xi)[0]
yid = res.sol(xi)[1]

plt.plot(xi, yi, label='f(x)');
plt.plot(xi, yid, label="f'(x)")
```

```
[17]: [<matplotlib.lines.Line2D at 0x1ab453c1130>]
```

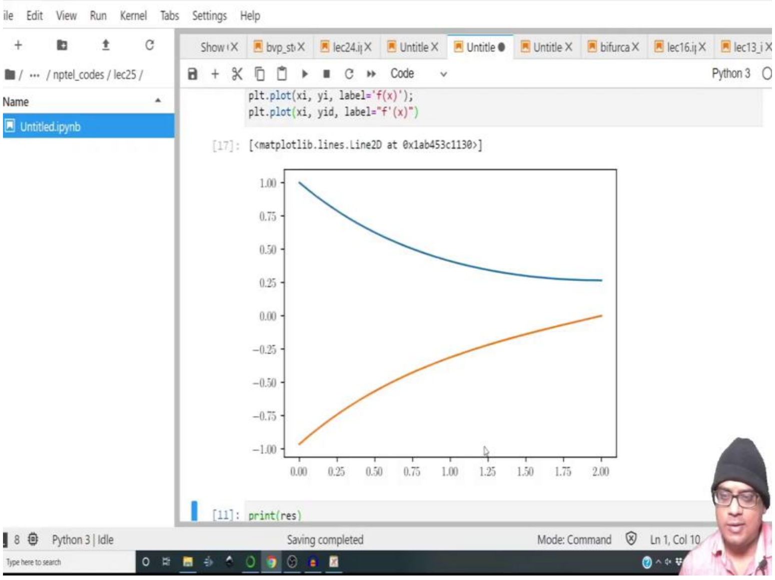
```
[11]: print(res)
```

```
       message: 'The algorithm converged to the desired accuracy.'
         niter: 1
             p: None
 rms_residuals: array([4.18885995e-05, 3.70210540e-05, 3.23734781e-05, 2.80680145e-05,
       2.42231906e-05, 2.09657631e-05, 1.84516896e-05, 1.68844152e-05,
       1.64997687e-05])
```

ile Edit View Run Kernel Tabs Settings Help
/ ··· / nptel_codes / lec25 /
Name
Untitled.ipynb
Show ‹ bvp_st lec24.i Untitle Untitle Untitle bifurca lec16.i lec13_i
Code
Python 3
8 Python 3 | Busy
Saving completed
Mode: Command
Ln 1, Col 10
Type here to search

Because let me use this instead. Yeah, this should work yeah.

(Refer Slide Time: 21:34)

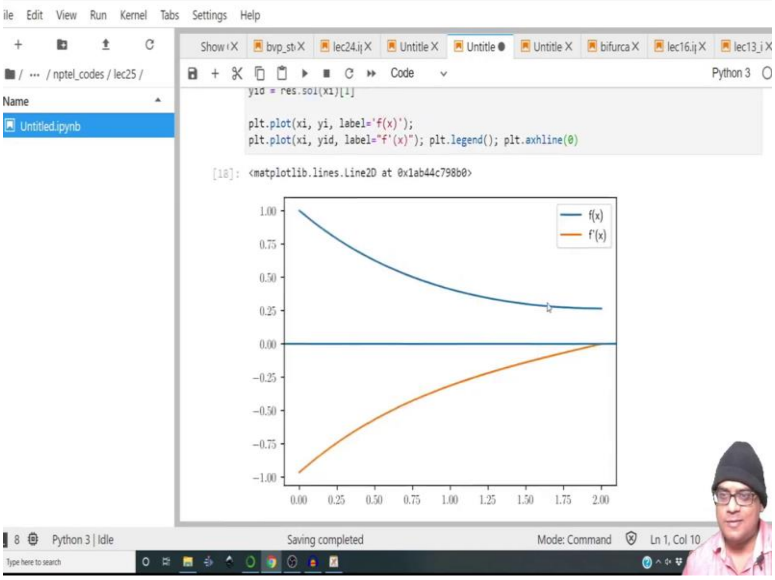


```
plt.plot(xi, yi, label='f(x)');
plt.plot(xi, yid, label="f'(x)")
```

```
[17]: [<matplotlib.lines.Line2D at 0x1ab453c1130>]
```

```
[11]: print(res)
```

(Refer Slide Time: 21:37)

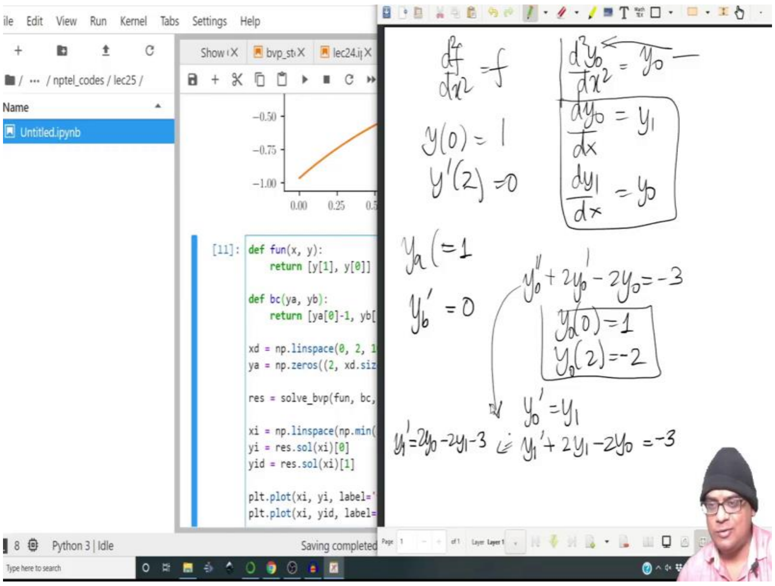


```
yid = res.sol(xi)[1]

plt.plot(xi, yi, label='f(x)');
plt.plot(xi, yid, label="f'(x)"); plt.legend(); plt.axhline(0)
```

```
[18]: <matplotlib.lines.Line2D at 0x1ab44c798b0>
```

So, then only just plt.legend and plt. axh line at 0. So, look at x = 2, this slope becomes 0. So, with the help of this, we are able to solve the problem I mean in a relatively short manner and I have I have showed you how to implement the boundary condition. The boundary condition has to be implemented as in the form of a residue ok. It sort of acts as an constraint to this optimization problem.

(Refer Slide Time: 22:31)



```
[11]: def fun(x, y):
          return [y[1], y[0]]

      def bc(ya, yb):
          return [ya[0]-1, yb[

      xd = np.linspace(0, 2, 1
      ya = np.zeros((2, xd.siz

      res = solve_bvp(fun, bc,

      xi = np.linspace(np.min(
      yi = res.sol(xi)[0]
      yid = res.sol(xi)[1]

      plt.plot(xi, yi, label='
      plt.plot(xi, yid, label=
```

$\frac{d^2f}{dx^2} = f$

$y(0) = 1$

$y'(2) = 0$

$\frac{d^2y_0}{dx^2} = y_0$

$\frac{dy_0}{dx} = y_1$

$\frac{dy_1}{dx} = y_0$

$y_a(= 1$

$y_b' = 0$

$y_0'' + 2y_0' - 2y_0 = -3$

$y_0(0) = 1$

$y_0(2) = -2$

$y_0' = y_1$

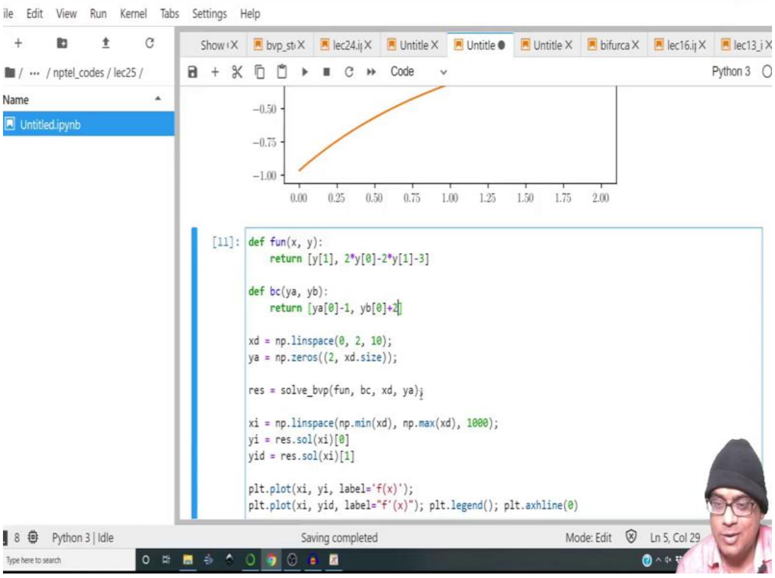
$y_1' = 2y_0 - 2y_1 - 3 \Leftarrow y_1' + 2y_1 - 2y_0 = -3$

So, yeah; let us do one more problem. That means, we do not need this anymore; let me place this over here, let us solve this problem. So, $y_0'' + 2y_0' - 2y_0 = -3$;

$y_0(0)=1$ and $y_0(2)=-2$. So, it is a second order differential equation, ordinary differential equation, two-point boundary value problem. And because we have developed the program before, it is quite easy for us to implement this.

So, let me make it y naught right and I can write $y_0'=y_1$, this becomes $y_1'+2y_1-2y_0=-3$ and these are still the boundary conditions alright. So, it will still return y 1; over here, it will return 2y0 - 2. So, essentially, this is $y_1'=2y_0-2y_1-3$.

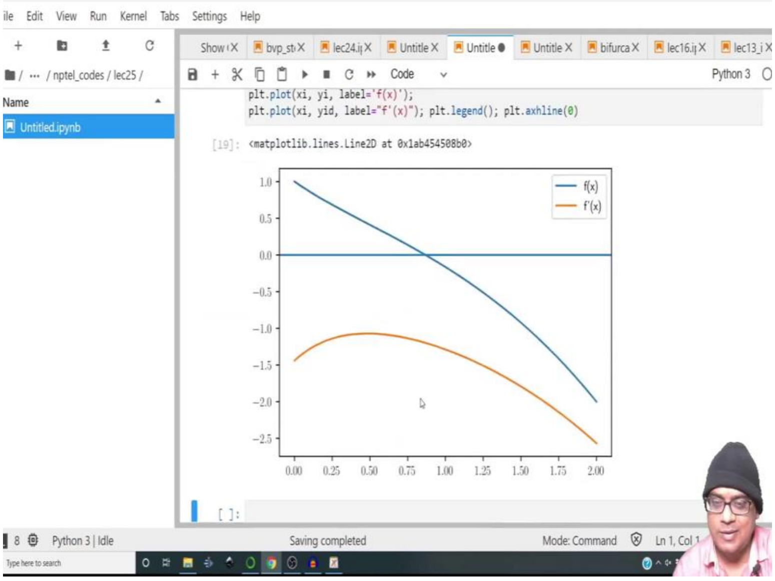
(Refer Slide Time: 23:46)



```
[11]: def fun(x, y):
          return [y[1], 2*y[0]-2*y[1]-3]

      def bc(ya, yb):
          return [ya[0]-1, yb[0]+2]

      xd = np.linspace(0, 2, 10);
      ya = np.zeros((2, xd.size));

      res = solve_bvp(fun, bc, xd, ya);

      xi = np.linspace(np.min(xd), np.max(xd), 1000);
      yi = res.sol(xi)[0]
      yid = res.sol(xi)[1]

      plt.plot(xi, yi, label='f(x)');
      plt.plot(xi, yid, label="f'(x)"); plt.legend(); plt.axhline(0)
```

So, its $y_1'=2y_0-2y_1-3$ and the boundary conditions are 1 and -2. So, it is already 1 and over here because it is not a Neumann condition, it is a Dirichlet condition, it was -2. So, this has to be +2. So, that the residue. So, the way of writing this boundary condition is y b + 2 = 0. So, y b becomes - 2 at the other boundary. The domain does span from 0 to 2? Yeah.

(Refer Slide Time: 24:20)

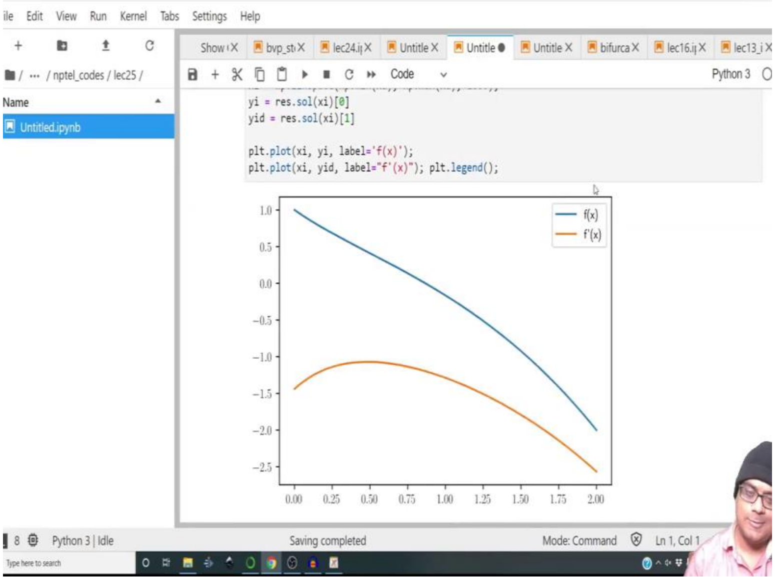


```
plt.plot(xi, yi, label='f(x)');
plt.plot(xi, yid, label="f'(x)"); plt.legend(); plt.axhline(0)
```

```
[19]: <matplotlib.lines.Line2D at 0x1ab454508b0>
```

So, yeah, let us solve this and see what happens. So, this is how the solution looks. We do not really need the horizontal line.

(Refer Slide Time: 24:28)



```
yi = res.sol(xi)[0]
yid = res.sol(xi)[1]

plt.plot(xi, yi, label='f(x)');
plt.plot(xi, yid, label="f'(x)"); plt.legend();
```

So, this is the solution of the problem and this is the derivative in case someone is interested. So, yeah, you can do problems like this; just have a look at the function reference for solve bvp and yeah, it clearly says it solves a first order system of ODEs like this subjected to two-point boundary conditions like this right.

So, things become much easier, when things are inbuilt. But even when they are not inbuilt, you can write your own program. It is not that difficult to implement really. It is just a matter of getting used to. So, with this, let us conclude this particular session. I will be back next time with regular perturbation. We will proceed to singular perturbation and let us see how we can progress from there.

With this, I wish you a good day and see you again next time. Bye.