Computational Aircraft Prototype Syntheses

Training Session 5
Aero Modeling: AVL and masstran

ESP v1.19

Marshall Galbraith
galbramc@mit.edu

Bob Haimes
haimes@mit.edu

Massachusetts Institute of Technology

John F. Dannenhoffer, III
jfdannen@syr.edu
Syracuse University

Overview

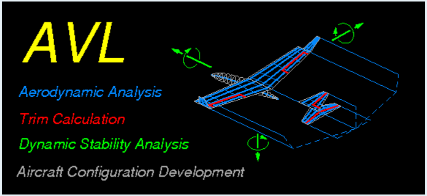
- AVL Overview
  - AVL Geometry Definition
  - Reference Quantities
- Control Surfaces and Stability Derivatives
- AVL Eigenmode Analysis
  - Pure AVL
  - AVL and masstran
- Linked Analysis Parameters
  - AVL and FRICTION
- Suggested Exercises

Athena Vortex Lattice – AVL

- Aerodynamic and flight-dynamic analysis of rigid aircraft

Extended Vortex-Lattice Model

- Aerodynamic Components
  - Lifting surfaces
  - Slender bodies
- Control deflections
  - Via normal-vector tilting
  - Leading edge or trailing edge flaps
- General freestream description
  - alpha,beta flow angles
  - p,q,r aircraft rotation
- Aerodynamic outputs
  - Forces and moments, in body or stability axes
  - Force and moment derivatives w.r.t. angles, rotations, controls



Athena Vortex Lattice – AVL cont.

Trim Calculation

- Operating variables
  - alpha,beta
  - p,q,r
  - control deflections
- Constraints
  - direct constraints on variables
  - indirect constraints via specified CL, moments
  - level or banked horizontal flight
  - steady pitch rate (looping) flight

Eigenmode analysis

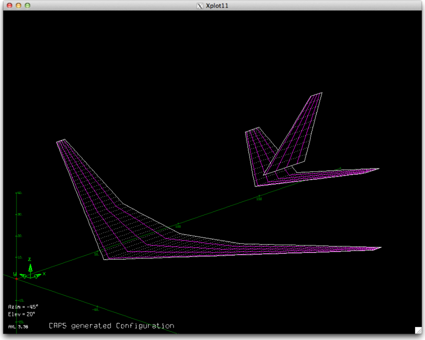
- Predicts flight stability characteristics
- Rigid-body, quasi-steady aero model
- Eigenvalue root progression with a parameter
- Display of eigenmode motion in real time

Vortex Lattice Geometry

Geometry specified with airfoil sections

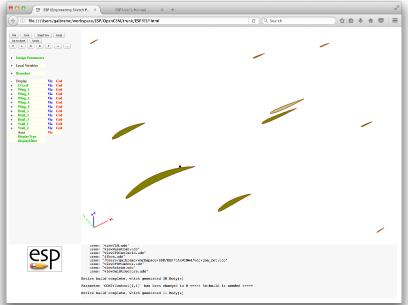
```
#--------------------------------------------------
SURFACE
WING
#Nchordwise  Cspace  Nspanwise  Sspace
1            1.0     16         -2.0
YDUPLICATE
0.0

SECTION
#Xle    Yle    Zle    Chord  Ainc  Nspanwise  Sspace
-0.25   0.     0.     1.000  0.    8          1.0
AIRFOIL
naca2412.dat

SECTION
#Xle    Yle    Zle    Chord  Ainc  Nspanwise  Sspace
-0.175  7.5    0.5    0.700  0.    0          0
AIRFOIL
naca0012.dat
```

VLM geometry is flat panels

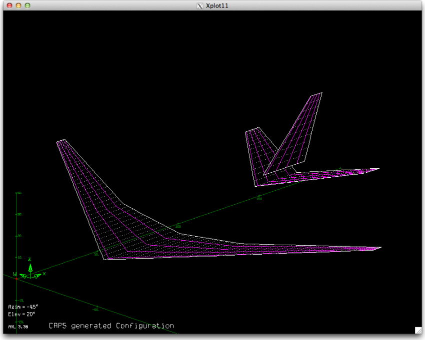


Vortex Lattice Geometry

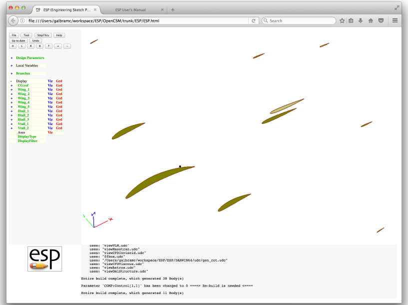
ESP geometry airfoil sections



VLM geometry is flat panels



ESP geometry airfoil sections



View Concept and VLM in ESP

---

cd training/CAPS/ESP
serveCSM transport.csm

VIEW:Concept	0
VIEW:VLM	1

Checking the Geometry with pyCAPS

session05/avl_1_TransportGeom.py

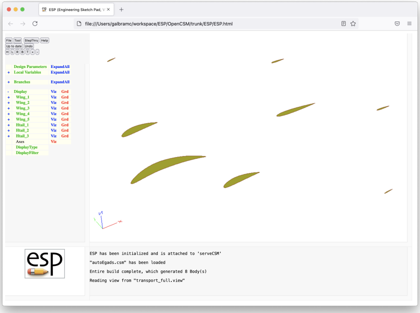
```
# Load geometry [.csm] file
filename = os.path.join("..","ESP","transport.csm")
print ("\n==> Loading geometry from file \""+filename+"\"...")
myProblem = pyCAPS.Problem(problemName = "workDir_1_TransportGeom",
                           capsFile = filename,
                           outLevel = 0)

# Alias the geometry
transport = myProblem.geometry

# Change to VLM view
transport.cfgpmtr.VIEW.Concept = 0
transport.cfgpmtr.VIEW.VLM     = 1

# view the geometry with ESP
print ("\n==> Viewing transport bodies...")
transport.view()

# Create AVL AIM
print ("\n==> Create AVL aim...")
avl = myProblem.analysis.create(aim  = "avlAIM",
                                name = "avl")

# view avl bodies with the capsViewer
print ("\n==> Viewing avl bodies...")
avl.geometry.view()
```

transport.view()

Checking the Geometry with pyCAPS

transport.view()

avl.view()



Checking the Geometry with pyCAPS

- Vertical tail is missing capsAIM attribute!
- ESP/viewVLM.udc must be fixed for session05 examples to work (uncomment line 218)

ESP/viewVLM.udc

```
    INTERSECT
        #ATTRIBUTE capsAIM         vlmAIMs
        ATTRIBUTE capsGroup       $Vtail
```

- Very rich input data set
  - Many geometric parameter
  - Multiple bodies
  - Many attributes on `BODY/FACE/EDGE/NODE`
- Not all error checking can be automated
- Significant user responsibility to check consistency
- Always check initial setup as much as possible

AVL AIM Documentation

Reference Quantities

AVL Input Header

```
!Sref     Cref     Bref
12.0      1.0      15.0
!Xref     Yref     Zref
0.0       0.0       0.0
```

ESP/viewVLM.udc

```
            RESTORE   WingOml
            INTERSECT
                ATTRIBUTE capsAIM            vlmAIMs
                ATTRIBUTE capsReferenceArea  wing:area
                ATTRIBUTE capsReferenceSpan  wing:span
                ATTRIBUTE capsReferenceChord wing:mac
                ATTRIBUTE capsReferenceX     wing:xroot+wing:mac/4
```

capsReference* attributes

Sref	capsReferenceArea	area for coefficients (C_L, C_D, C_m, etc)
Cref	capsReferenceChord	chord for pitching moment (C_m)
Bref	capsReferenceSpan	span for roll,yaw moments (C_l, C_n)
Xref	capsReferenceX	location for moments, rotation rates

capsReference* attributes on one or more bodies (consistent)

Overview

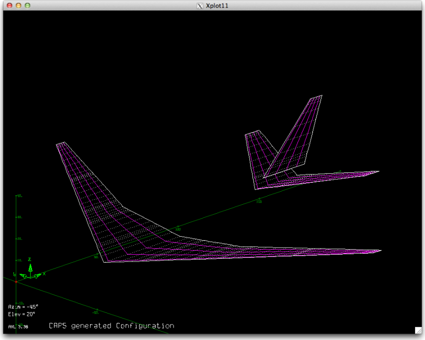
- AVL Overview
  - AVL Geometry Definition
  - Reference Quantities
- Control Surfaces and Stability Derivatives
- AVL Eigenmode Analysis
  - Pure AVL
  - AVL and masstran
- Linked Analysis Parameters
  - AVL and FRICTION
- Suggested Exercises

Vortex Lattice Geometry with Controls

- Controls specified with airfoil sections
- Airfoil interpolation?

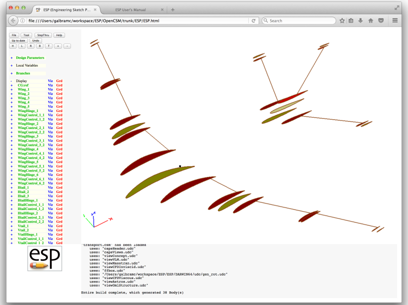
```
#-----------------------------------------------------
SURFACE
STAB
#Nchordwise  Cspace  Nspanwise  Sspace
1            1.0     7          -2.0
YDUPLICATE
0.0

SECTION
#Xle    Yle    Zle     Chord   Ainc  Nspanwise  Sspace
6.0    0.     0.0     0.4     0.    7          -1.25
CONTROL
#name     gain Xhinge  XYZhvec   SgnDup
elevator  1.0  0.0     0. 1. 0.   1

SECTION
#Xle    Yle    Zle     Chord   Ainc  Nspanwise  Sspace
-0.075  2.00   0.0     0.3     0.    0          0
CONTROL
#name     gain Xhinge  XYZhvec   SgnDup
elevator  1.0  0.0     0. 1. 0.   1
```

Mesh clustering around controls

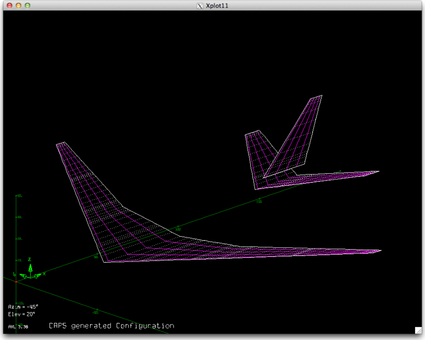


Vortex Lattice Geometry with Controls

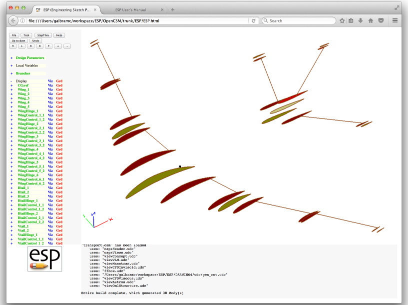
ESP control airfoil sections (red)



Mesh clustering around controls



ESP control airfoil sections (red)



View Concept and VLM in ESP

---

cd training/CAPS/ESP
serveCSM transport.csm

VIEW:Concept	0
VIEW:VLM	1
COMP:Control	1

vlmControl_"Name" Attribute

- vlmControl_"Name" specifies a section with a control surface
  - "Name" is the name of the control surface
  - Value is chord fraction of hinge line

session05/avlPlaneVanilla.csm

Wing Right Tip Airfoil

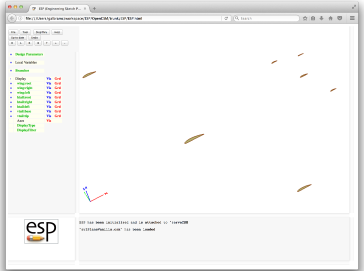
```
TRANSLATE wing:xtip    -wing:span/2    wing:ztip
SELECT body
    ATTRIBUTE vlmControl_AileronRight 0.8  #Hinge line 80% chord
    ATTRIBUTE capsGroup               $Wing
```

Wing Root Airfoil

```
TRANSLATE wing:xroot    0    wing:zroot
SELECT body
    ATTRIBUTE vlmControl_AileronRight 0.8  #Hinge line 80% chord
    ATTRIBUTE vlmControl_AileronLeft  0.8  #Hinge line 80% chord
    ATTRIBUTE capsGroup               $Wing
```

Wing Left Tip Airfoil

```
TRANSLATE wing:xtip     wing:span/2    wing:ztip
SELECT body
    ATTRIBUTE vlmControl_AileronLeft 0.8  #Hinge line 80% chord
    ATTRIBUTE capsGroup              $Wing
```



vlmControl_"Name" Attribute

- vlmControl_"Name" specifies a section with a control surface
  - "Name" is the name of the control surface
  - Value is chord fraction of hinge line

session05/avlPlaneVanilla.csm

```
UDPRIM    naca  Thickness wing:thick  Camber wing:camber
SCALE     wing:croot
ROTATEX   90       0         0
TRANSLATE wing:xroot    0    wing:zroot
SELECT body
    ATTRIBUTE vlmControl_AileronRight 0.8  #Hinge line 80% chord
    ATTRIBUTE vlmControl_AileronLeft  0.8  #Hinge line 80% chord
    ATTRIBUTE capsGroup               $Wing
```

session05/avl_2_PlaneVanillaControl.py

```
# Set control surface parameters
aileronLeft  = {"deflectionAngle" : -25.0}
aileronRight = {"deflectionAngle" :  25.0}
elevator     = {"deflectionAngle" :  5.0}
rudder       = {"deflectionAngle" : -2.0}

avl.input.AVL_Control = {"AileronLeft"  : aileronLeft ,
                         "AileronRight": aileronRight,
                         "Elevator"    : elevator    ,
```

Transport Controls

ESP/transport.csm

```
# wing hinge lines
DIMENSION wing:hinge         6 9   # ymin                  ymax
#                            theta   x/c   y/span  z/t      x/c   y/span z/t    gap   grp
DESPMTR   wing:hinge        "-10.0;  0.75; -0.98;  0.50;   0.75; -0.70; 0.50;  0.25; 1;  \ left aileron
                             +10.0;  0.75; -0.69;  0.00;   0.75; -0.43; 0.00;  0.25; 2;  \ left oflap
                             +15.0;  0.85; -0.33;  0.00;   0.90; -0.14; 0.00;  0.25; 3;  \ left iflap
                             +15.0;  0.90;  0.14;  0.00;   0.85;  0.33; 0.00;  0.25; 3;  \ rite iflap
                             +10.0;  0.75;  0.43;  0.00;   0.75;  0.69; 0.00;  0.25; 2;  \ rite oflap
                             +10.0;  0.75;  0.70;  0.50;   0.75;  0.98; 0.50;  0.25; 4"  # rite aileron
```

ESP/viewVLM.udc

```
                    ATTRIBUTE capsGroup                          $Wing
                    ATTRIBUTE capsDiscipline                     $Aerodynamic
                    ATTRIBUTE _name                             !$WingControl_+ihinge+$_1
                    ATTRIBUTE !$vlmControl_WingControl+tagIndex xoverc1
```

session05/avl_3_TransportControl.py

```python
controls = {}

hinge = transport.despmtr.wing.hinge
for i in range(len(hinge)):
    controls["WingControl_"+str(int(hinge[i][8]))] = {"deflectionAngle": hinge[i][0]}
```

Transport Stability Derivatives

- Stability derivatives available as analysis outputs

session05/avl_3_TransportControl.py

```
print ("\n==> Some stability derivatives")
print ("--> CLa  =", avl.output["CLa"].value ) # - Alpha
print ("--> CLb  =", avl.output["CLb"].value ) # - Beta
print ("--> CLp' =", avl.output["CLp'"].value) # - Roll rate
print ("--> CLq' =", avl.output["CLq'"].value) # - Pitch rate
print ("--> CLr' =", avl.output["CLr'"].value) # - Yaw rate

# Get neutral point, CG and MAC
Xnp = avl.output.Xnp
Xcg = avl.output.Xcg
MAC = transport.outpmtr.wing.mac

StaticMargin = (Xnp - Xcg)/MAC
print ("--> Xcg = ", Xcg)
print ("--> Xnp = ", Xnp)
print ("--> StaticMargin = ", StaticMargin)

ControlStability = avl.output.ControlStability
for key in ControlStability.keys():
    print("-->", key, ControlStability[key])
```

Overview

- AVL Overview
  - AVL Geometry Definition
  - Reference Quantities
- Control Surfaces and Stability Derivatives
- AVL Eigenmode Analysis
  - Pure AVL
  - AVL and masstran
- Linked Analysis Parameters
  - AVL and FRICTION
- Suggested Exercises

Eigenmode analysis

- Stable configuration: all negative real Eigen values
- Requires realistic:
  - Configuration
  - Mass, CG, and inertia data
  - Flight conditions
- AVL Eigenmode analysis requires dimensional units
  - Geometry units defined with `capsLength` `BODY` attribute

ESP/transport.csm

```
# Define length units of the geometry
ATTRIBUTE capsLength      $ft
```

Defining the unit system

- Define units consistent with udunits2 convection
- Define the unit system
- Analysis unitSystem triggers inputs/outputs with units

session05/avl_4_TransportEigen.py

```
# Define units
m    = pyCAPS.Unit("meter")
kg   = pyCAPS.Unit("kg")
s    = pyCAPS.Unit("s")
K    = pyCAPS.Unit("Kelvin")
deg  = pyCAPS.Unit("degree")
ft   = pyCAPS.Unit("ft")
slug = pyCAPS.Unit("slug")
lb   = pyCAPS.Unit("lb")

unitSystem={"mass":kg, "length":m, "time":s, "temperature":K}
```

```
avl = myProblem.analysis.create(aim = "avlAIM",
                                name = "avl",
                                unitSystem = unitSystem)
```

```
# Set analysis specific variables
avl.input.Mach  = 0.5
avl.input.Alpha = 1.5*deg
avl.input.Beta  = 0.0*deg
```

Eigenmode Analysis with Static Components

- Specifying Eigenmode Analysis dimensional inputs

session05/avl_4_TransportEigen.py

```
# Inspired by the b737.mass avl example file
#                   mass           CGx  CGy CGz        Ixx     Iyy     Izz
cockpit   = {"mass": 3000 * lb, "CG":[  8,  0,  5] * ft, "massInertia":[0.    ,  0.    ,  0.    ] * lb*ft**2}
wing      = {"mass":19420 * lb, "CG":[ 78,  0, -1] * ft, "massInertia":[8.0e6,  0.1e6,  8.1e6] * lb*ft**2}
fuselage  = {"mass":33720 * lb, "CG":[105,  0,  2] * ft, "massInertia":[0.7e6, 18.9e6, 19.6e6] * lb*ft**2}
tailcone  = {"mass":  310 * lb, "CG":[145,  0,  0] * ft, "massInertia":[0.    ,  0.    ,  0.    ] * lb*ft**2}
Htail     = {"mass":  528 * lb, "CG":[160,  0,  2] * ft, "massInertia":[0.0e6,  0.0e6,  0.0e6] * lb*ft**2}
Vtail     = {"mass":  616 * lb, "CG":[100,  0,  8] * ft, "massInertia":[0.1e6,  0.0e6,  0.1e6] * lb*ft**2}
Main_gear = {"mass": 4500 * lb, "CG":[ 76,  0, -4] * ft, "massInertia":[0.5e6,  0.0   ,  0.5e6] * lb*ft**2}
Nose_gear = {"mass": 1250 * lb, "CG":[ 36,  0, -5] * ft, "massInertia":[0.    ,  0.    ,  0.    ] * lb*ft**2}

avl.input.MassProp = {"cockpit"  : cockpit  ,
                      "wing"     : wing     ,
                      "fuselage" : fuselage ,
                      "tailcone" : tailcone ,
                      "Htail"    : Htail    ,
                      "Vtail"    : Vtail    ,
                      "Main_gear": Main_gear,
                      "Nose_gear": Nose_gear}
avl.input.Gravity  = 32.18 * ft/s**2
avl.input.Density  = 0.002378 * slug/ft**3
avl.input.Velocity = 250.0 * m/s
```

Eigenmode Analysis with masstran

- Check bodies passed to avl and masstran

session05/avl_masstran_5_Geom.py

```python
transport.cfgpmtr.VIEW.Concept      = 0
transport.cfgpmtr.VIEW.VLM          = 1
transport.cfgpmtr.VIEW.OmlStructure = 1

# Enable fuselage and lifting surfaces
transport.cfgpmtr.COMP.Wing    = 1
transport.cfgpmtr.COMP.Fuse    = 1
transport.cfgpmtr.COMP.Htail   = 1
transport.cfgpmtr.COMP.Vtail   = 1
transport.cfgpmtr.COMP.Control = 0

# Create AVL AIM
avl = myProblem.analysis.create(aim = "avlAIM",
                                name = "avl")

print ("\n==> Vewing AVL geometry")
avl.geometry.view()

masstran = myProblem.analysis.create(aim = "masstranAIM",
                                     name = "masstran")

print ("\n==> Vewing Masstran geometry")
masstran.geometry.view()
```

Eigenmode Analysis with masstran

- Get mass properties from masstran

session05/avl_masstran_6_Eigen.py

```
# Set property
shell  = {"propertyType"      : "Shell",
          "material"          : "Unobtainium",
          "membraneThickness" : 0.02 * ft}

masstran.input.Property = {"fuseSkin" : shell,
                           "wingSkin" : shell,
                           "htailSkin": shell,
                           "vtailSkin": shell}

print ("\n==> Computing mass properties...")
masstran.preAnalysis()
masstran.postAnalysis()

aircraft_mass = masstran.output.Mass
aircraft_CG   = masstran.output.CG
aircraft_I    = masstran.output.I_Vector

aircraft_skin = {"mass":aircraft_mass, "CG":aircraft_CG, "massInertia":aircraft_I}

print ("\n==> Computed mass properties...")
print ("--> aircraft_skin = ", aircraft_skin)
```

- Pass mass properties to AVL

session05/avl_masstran_6_Eigen.py

```
#                    mass            CGx  CGy  CGz                     Ixx    Iyy  Izz
cockpit   = {"mass": 3000 * lb, "CG":[  8,  0,  5] * ft, "massInertia":[0.   ,  0.   ,  0.   ] * lb*ft**2}
Main_gear = {"mass": 4500 * lb, "CG":[ 76,  0, -4] * ft, "massInertia":[0.5e6,  0.0  ,  0.5e6] * lb*ft**2}
Nose_gear = {"mass": 1250 * lb, "CG":[ 36,  0, -5] * ft, "massInertia":[0.   ,  0.   ,  0.   ] * lb*ft**2}

avl.input.MassProp = {"aircraft_skin": aircraft_skin,
                      "cockpit"      : cockpit      ,
                      "Main_gear"    : Main_gear    ,
                      "Nose_gear"    : Nose_gear    }
avl.input.Gravity  = 32.18 * ft/s**2
avl.input.Density  = 0.002378 * slug/ft**3
avl.input.Velocity = 250.0 * m/s
```

Overview

- AVL Overview
  - AVL Geometry Definition
  - Reference Quantities
- Control Surfaces and Stability Derivatives
- AVL Eigenmode Analysis
  - Pure AVL
  - AVL and masstran
- Linked Analysis Parameters
  - AVL and FRICTION
- Suggested Exercises

FRICTION

- Virginia Tech Aerodynamics and Design Software Collection
- Estimates subsonic skin friction and form drag
- Suitable for aircraft preliminary design

- Link FRICTION output drag to AVL input profile drag
- Couples DIRTY/CLEAN process
- Automatically transfers data
- Data “shape” and “type” must be the same

session05/avl_friction_7_Link.py

```
avl.input["CDp"].link(friction.output["CDtotal"])
```

Common Analysis Parameters

- Both FRICTION and AVL have "Mach" input (must be consistent)
- Create a common analysis parameter and link to AVL/FRICTION input

session05/avl_friction_7_Link.py

```
# Create shared analysis input parameter
mach = myProblem.parameter.create("Mach", 0.1)
```

```
friction.input["Mach"].link(mach)
```

```
avl.input["Mach"].link(mach)
```

```
machRange = [mach.value + i*0.1 for i in range(7)]
for M in machRange:
    # Set mission parameters - AIMs will be updated automatically due to link
    mach.value = M
```

Suggested Exercises

Multiple Shells

- Create multiple materials and shell properties for the transport components in avl_masstran_6_Eigen.py

Multiple AIMs

- Create multiple masstran AIM instances for the transport components in avl_masstran_6_Eigen.py
  - Note: viewOmlStructure has same `capsIntent` as F-118

Main Gear

- Use wing:xroot and wing:mac to position the main gear CGx in avl_masstran_6_Eigen.py as a fraction of wing:mac downstream of wing:xroot

Suggested Exercises Cont.

Stable Transport

- Resize tail and modify mass properties of avl_masstran_6_Eigen.py to make a stable transport (all negative real Eigen values)
  - See session05/transport_Htail.py as an example of sweeping through tail size

- Create your own (optionally share it `galbramc@mit.edu`)